

PROYECTO DE GRADO – SISTEMAS DE WORKFLOW.

Javier Apaulaza, José Casarini, Rodrigo Recoba

Tutor: Guillermo Calderón
Usuario Responsable: Daniel Gómez

Instituto de Computación
Facultad de Ingeniería
Universidad de la República Oriental del Uruguay.

Marzo 2004

Resumen.

En el presente trabajo abordamos la construcción de un sistema de workflow. El objetivo del mismo es desarrollar una herramienta que permita modelar fácilmente los flujos de trabajo y su posterior ejecución, automatizando al máximo los procesos.

Podemos definir al Workflow como un conjunto de métodos y tecnologías que nos ofrecen las facilidades para modelar y gestionar los diversos procesos que ocurren dentro de una empresa, de forma de controlar, automatizar y agilizar los mismos.

Hemos desarrollado un Sistema de Workflow implementado íntegramente en la nueva plataforma Microsoft.NET, utilizando su nuevo lenguaje C# y SQL Server 2000 como base de datos. Dicho sistema fue realizado para la empresa INFOCORP, la cual pretendía obtener un prototipo de WorkFlow que pueda constituirse como un producto independiente así como también sirva como base para todos los productos (implementaciones prácticas del WorkFlow) que la empresa ofrece, transformándose en un framework que permita el desarrollo de aplicaciones de forma rápida y estandarizada, utilizando tecnología de última generación.

Para poder llevar a cabo el proyecto hubo que realizar diversos estudios referentes a los sistemas de workflow, revisando diversos estándares, principalmente los establecidos por la WfMC (Workflow Management Coalition). También se estudiaron diversos productos existentes en el mercado como ser XData y QFlow, los cuales fueron de gran ayuda a la hora de detallar las principales funcionalidades de los sistemas de Workflow. De este primer relevamiento surgió la necesidad de estudiar, además del lenguaje y plataforma utilizados, diversas tecnologías, como ser: Lenguajes de Definición de Procesos (XPDL – XML, XSL, XSLT), mecanismos de comunicación (MSMQ, Remoting, Web Services), Librerías de .NET como ser ADO.NET, ASP.NET, bases de datos Microsoft SQL Server y servidores Web Internet Information Server 5.0.

Cómo resultado de las investigaciones realizadas se construyó un prototipo de un sistema de workflow que permite modelar fácilmente un flujo de trabajo o proceso de negocio, y su posterior ejecución. El mismo brinda la posibilidad de crear un proceso de negocio que luego podrá ser iniciado, cancelado, pausado y reiniciado. Además se brindan las funcionalidades mas comunes para la definición de procesos, como ser: enviar mail, fax, invocar Web services, invocar Assemblies y ejecutar scripts desarrollados utilizando VBScript o JScript, También se brindan las funcionalidades necesarias para monitorear el estado de un proceso y ver su historial, es decir, las etapas por las cuales fue pasando y los resultados obtenidos. Dicho prototipo fue de satisfacción para la Infocorp y fue utilizado como base para su producto de workflow.

ÍNDICE

1	INTRODUCCIÓN.....	3
1.1	Estado de Arte.....	3
1.2	Introducción a los Sistemas de Workflow.....	7
1.3	Descripción del proyecto.....	11
1.4	Contexto del Proyecto.....	11
1.5	Síntesis del informe.....	12
2	DESCRIPCIÓN DEL PROBLEMA.....	13
2.1	Estructura Genérica de un Producto de Workflow.....	15
3	ALCANCE DEL PROYECTO.....	19
3.1	Funcionalidades a Desarrollar.....	19
4	PLANIFICACIÓN DEL PROYECTO.....	21
4.1	Cronograma.....	21
4.2	Riesgos.....	21
5	SOLUCIÓN AL PROBLEMA.....	22
5.1	Lenguaje y plataforma utilizados.....	22
5.2	Arquitectura.....	24
5.3	Componentes principales.....	27
5.4	Diseño de los componentes.....	28
6	CASO DE USO – SOLICITUD DE LICENCIA.....	42
6.1	Definición de esquemas.....	42
6.2	Definición de Procesos.....	44
6.3	Transformación Final.....	68
7	TRABAJO A FUTURO.....	69
7.1	Extensión del Prototipo.....	69
7.2	Sistemas que se montarán sobre el sistema de workflow.....	70
8	CONCLUSIONES:.....	71
9	Referencias:.....	72

1 INTRODUCCIÓN.

En este capítulo se presentará el estado actual de los sistemas de workflow, señalando el estado del arte y brindando un marco teórico que permitirá comprender los sistemas de workflow y el resto del documento. También se describe brevemente el proyecto y el contexto en el cual se realizó, finalizando con una síntesis que explica como se lee el resto del documento.

1.1 Estado de Arte

Si bien las personas somos buenas para tomar decisiones, innovar e identificar hechos inesperados, no tenemos la misma capacidad para realizar tareas tales como: tener presentes cronogramas, fechas o documentos, realizar búsquedas o seguimientos, verificar que los trabajos culminados pasen a su siguiente etapa respetando una secuencia definida, por ejemplo. Siendo estas tareas de vital importancia para las organizaciones, es que surgió la necesidad de automatizarlas, dando origen así al concepto de Workflow.

Al igual que la evolución de la informática en general, la evolución del Workflow está ligada con el cambio en los objetivos de cada época. Resumiendo la evolución de la informática en las últimas cuatro décadas, se puede ver como han ido cambiando los objetivos a seguir en cada una de ellas. En la década de los 60' y 70' el gran objetivo era resolver grandes cantidades de cálculo de manera eficiente. En los 80' se buscaba mejorar el manejo y administración de las bases de datos y en los 90' surge la necesidad de entender y manejar eficientemente los flujos de trabajo (Workflow), de manera de poder obtener el mayor provecho. Mirando la actuación del Workflow dentro de estas tres etapas, se puede identificar lo que sería un Workflow Manual en la primera etapa, el Workflow Automatizado dentro de la segunda, y lo que ofrece un Workflow en la actualidad.

En el primer caso se puede ver que antes de que la informática se integrara al trabajo cotidiano, éste era realizado manualmente combinando toda la información en distintas carpetas. En este ambiente era bastante difícil determinar el estado de una determinada carpeta, así como también determinar el proceso a seguir. Se manejaban grandes cantidades de documentos en forma manual, con los consiguientes errores humanos que traían aparejados dichos manejos. Por esto es posible identificar un Workflow Manual inmerso en las tareas cotidianas de esta época. Surge la necesidad de reemplazar las actividades manuales por actividades automáticas, es decir, se busca tener un mayor control y coordinación sobre toda la información que se maneja para llevar a cabo las tareas de las organizaciones.

En la década del 80', se puede apreciar la existencia de diversos sistemas de información necesaria para llevar a cabo la producción de las empresas. Se ha logrado automatizar ciertas tareas, que antes se realizaban manualmente. Por esto se habla de un Workflow Automatizado. A fines de esta década, se busca mejorar el flujo de la información, el desafío pasa a ser obtener la información rápida y eficientemente, surgen las necesidades de incrementar la eficiencia, optimizar la productividad, acortar los tiempos de procesos, tener un control sobre éstos, así como también de reducir los costos y mejorar la gestión. Todo esto como consecuencia del incremento de la competitividad y de la exigencia de mejores productos, dentro de un mercado de alta competencia que avanza a gran velocidad.

Hoy, un paso más allá, el objetivo es resolver eficientemente el Workflow. Actualmente existe una proliferación de diversos mecanismos de intercambio de información que facilitan el manejo general de los flujos de información. Las metas son similares a las de épocas anteriores, pero el punto de partida, las asunciones, y el impacto son distintos. Dentro de la evolución actual del Workflow como tecnología, se identifica la evolución y creación de ciertos productos que acompañan al mismo como ser:

- Procesamiento de imágenes,
- Administración de documentos,
- Aplicaciones basadas en transacciones,
- Procesamiento de formularios,
- Correo electrónico, etc. [13], [14], [15].

En la actualidad existen diversas empresas que ofrecen sistemas de workflow, entre las cuales se han estudiado y se destacan los productos XFlow [16] y QFlow [18], describiendo a continuación cada uno de ellos.

Q-flow es un completo conjunto de herramientas informáticas que proveen una solución integral de Workflow. Particularmente desarrollado para las áreas de Operaciones de las Empresas, cuya función central conlleva el desarrollo y establecimiento de procedimientos eficientes. Q-flow permite delinear en forma gráfica los procesos e implementar en forma casi inmediata la automatización de los mismos, ya que no requiere otro componente del lado del usuario más que el acceso a la mensajería (e-mail) de la empresa y el Navegador de Internet. Al ofrecer un camino rápido para la implementación de una sólida solución de automatización de procesos, integrándose completamente a la organización existente, Q-flow evita que los procesos se desvíen de su curso (tanto en forma como en tiempo). Su habilidad de monitoreo y gestión de los procesos históricos permite llevar un control sistemático de los tiempos insumidos en las más diversas actividades de la empresa, complementado con una amplia variedad de estadísticas acerca de procesos y usuarios. Provee comunicación bi-direccional con los sistemas informáticos de la empresa, habilitando así la posibilidad de iniciar o participar en flujos desde otras aplicaciones, así como también que un flujo cargue datos en bases existentes, lance aplicaciones, o reutilice componentes desarrollados por la empresa o terceros dentro de los Workflows diseñados con Q-flow.

Se enumeraran a continuación los componentes de Q-flow:

- Componentes Básicos.
 - Motor de Workflow.
 - Acceso Web.
- Herramientas de Integración.
 - Integrador de Aplicaciones.
 - Diseñador de Dominios.
- Herramienta de Diseños.
 - Diseñador de Templates.
 - Diseñador de Formularios.
- Herramientas de Administración.
 - Administrador de Seguridad.
 - Administrador de la estructura Organizacional.
- Herramientas de Análisis.
 - Generador de Cubos OLAP.
 - Pack de estadísticas.
- Componentes para Desarrolladores.
 - Pack de objetos COM para acceso, interacción con y extensión de Q-flow.

Las herramientas de diseño de Q-flow permiten definir, en forma gráfica, el proceso empresarial a automatizar, expresándolo en forma de tareas genéricas (preguntas, notificaciones, evaluaciones, puntos de control y de finalización, etc). Estas herramientas permiten también definir los roles propios del proceso (Ej.: autorizador, revisor, etc.), así como la información estructurada (o no estructurada) que se requerirán en el proceso (número de factura, moneda, importe, documento de especificaciones, etc).

Todos los diseños realizados en Q-flow son intercambiables entre distintas instalaciones de Q-flow por medio de archivos XML, ya que todo elemento diseñado en Q-flow puede exportarse a (e importarse desde) archivos XML. Por otro lado, el diseñador gráfico de procesos permite "dibujar" en forma totalmente gráfica los formularios que se utilizarán en el ingreso de información en los diversos pasos (Q-flow genera automáticamente los formularios y estos pueden ser rediseñados o generados totalmente desde cero según las necesidades del usuario).

Xflow es una herramienta desarrollada luego de varios años de experiencia en la implementación de soluciones de Workflow de Documentos. Permite definir gráficamente las "Rutas" que deben recorrer cada documento y las autorizaciones necesarias en un entorno Intranet o Extranet. De esta manera, se logra un mejor control y seguimiento de los procesos de negocio aumentando la productividad.

XFlow esta conformado por diferentes componentes, los cuales brindan funcionalidades específicas enumeradas a continuación:

Administrador de XFlow

- Diseño de diagramas de flujo
- Diseño de rutas (grafo de estados, grafo de páginas)
- Definición de permisos
- Edición de páginas
- Edición de vistas y búsquedas
- Enlace con la base de datos
- Definición de eventos de tiempo
- Definición de parámetros del sistema
- Definición de formularios a imprimir

Administrador de Entidades

- Administración de Entidades, subentidades y relaciones
- Impresión de la estructura de las entidades definidas
- Generación de estructuras en la base de datos

Administrador de Seguridad

- Definición de usuarios
- Definición de Tipos de Roles
- Definición de Roles
- Definición de Diagramas Jerárquicos
- Definición de Grupos
- Definición de Directivas de seguridad
- Definición de campos adicionales de usuarios.

Módulo de Conexión

- Ingreso de usuarios al sistema
- Control de horarios de ingreso
- Suplencias
- Control de directivas de seguridad de los usuarios
- Generación de auditoria de ingresos al sistema de usuarios

Servicio EBD

- Ejecución de acciones de enlace con la base de datos
 - Encoladas (en cola de mensajes)

Servicio Agenda

- Control de tiempo de los documentos en espera de ruteo
 - En forma Absoluta (Agenda)
 - En forma relativa (Timer)

Motor Xflow

- Ruteo de documentos
- Suplencias
- Mensajería
- Cambios de Actores/Lectores
- Agregado de Actores/Lectores

Mencionaremos a continuación algunas de las ventajas de Xflow:

- El Administrador de Procesos permite definir gráficamente la "Ruta" de cada documento y los "Estados" por los que va pasando.
- El Motor de Ruteo de Xflow se encarga de analizar y rutear automáticamente los documentos, ejecutando las acciones y notificaciones configuradas. Además se pueden personalizar acciones y tareas para adaptarse a los requerimientos específicos de su negocio
- Xflow permite medir los procesos para evaluarlos y mejorarlos. Se puede saber en cualquier momento quienes han intervenido en un documento y cual es su estado. Al mismo tiempo almacena automáticamente la historia de los datos ingresados, y las modificaciones efectuadas en cada paso.
- Con el Diseñador Inteligente de Páginas y Formularios se pueden crear fácilmente formularios específicos para su negocio y adaptarlos rápidamente según las necesidades.
- A cada documento se le pueden anexar planillas Excel, documentos Word, imágenes, y observaciones en cada estado.
- El Administrador de Entidades actúa como interfase entre SQL y Xflow facilitando la administración de bases de datos.
- El Administrador de Seguridad permite definir usuarios, roles, cargos, ubicación, jerarquías, grupos, suplencias o relaciones, para establecer distintos niveles de operatividad..

Los productos anteriormente detallados, dan una idea general sobre las funcionalidades que brindan en la actualidad los productos de workflow que se encuentran en el mercado. De esta forma, se puede discriminar con mayor facilidad cuales son los componentes existentes en la teoría, que realmente son de utilidad, y de que forma se implementan en la práctica.

1.2 Introducción a los Sistemas de Workflow

Un Workflow puede ser descrito como la administración del flujo de información y control de la misma en un proceso de negocio. Se puede definir un proceso de negocio como: El 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. [4]

Todas las organizaciones tienen numerosos procesos de negocios y algunos de estos procesos son claves para la supervivencia y su crecimiento. Administrando estos procesos más eficientemente, las compañías pueden reducir sus costos y mejorar sus productos, y servicios al consumidor. Ejemplos de procesos de negocios puede ser: órdenes de compra, solicitud de vacaciones, aprobación de cronogramas de desarrollo, procesos de selección de personal, etc., etc.

Un sistema de gestión de workflow provee automatización a los procesos de negocio, mediante la administración de la secuencia de actividades de trabajo, y la invocación de los recursos asociados a cada una de ellas. Se puede definir un sistema de gestión de workflow como: Un sistema que define, administra y ejecuta completamente workflows, a través de la ejecución de software cuyo orden de ejecución es dirigido por una representación computacional de la lógica del workflow [1].

Son varias las definiciones que se han hecho del término "Workflow". A nuestro entender una de las definiciones más adecuadas es:

Un Workflow es una estructura aplicada al movimiento de la información, para mejorar los resultados de los procesos de negocio por medio de la administración y coordinación de las actividades que desempeñan las personas involucradas en estos procesos de negocio.

[Workflow 94]

Independientemente de esta definición, existen tres conceptos claves dentro de lo que concierne a los workflows. Estos conceptos son actividad, coordinación y recursos. Un Workflow toma decisiones basadas en las condiciones del flujo determinadas durante el diseño, notifica a los recursos involucrados en el proceso, establece el trabajo que los mismos deben desarrollar, el estado en el que se encuentra una actividad, etc.

Uno de los principales conceptos en los sistemas de Workflow son las personas. Para lograr el éxito en el desarrollo se debe conseguir que las personas realicen sus tareas y las notifiquen en tiempo y forma. Por ejemplo, una tarea en un Workflow podría ser notificar (vía mail) a una persona que debe realizar cierta actividad. Dicha actividad puede ser manual (por ejemplo colocar materia prima en una determinada maquinaria) por lo que su finalización, algunas veces, sería dada manualmente por el usuario. Si este no lo hace ni bien culmina la tarea, el Workflow podría estar detenido esperando por la finalización de la misma hasta que este así se lo comunique. Como se ve, el concepto persona es importante para la implantación de un sistema de Workflow, ya que para el éxito del mismo no alcanza con que se realice la tarea, sino que también se precisa que los usuarios lo acepten y adopten como parte del desarrollo de sus tareas diarias.

Los workflows facilitan la coordinación del trabajo entorno a las capacidades de compartir información. Por ejemplo, suponiendo que se está procesando una orden de compra, el Workflow conoce a quien (persona o proceso) debe enviar la misma para que la valide, verifique, etc. Es decir, el Workflow conoce cada una de las etapas por las que debe pasar un proceso, a quien asignarle cada tarea del mismo, y además es el encargado de automatizar el pasaje de una etapa a otra.

Como ya se ha definido, un sistema de Workflow intenta automatizar total o parcialmente un proceso de negocio. Para esto, debe interactuar con diferentes procesos, en diferentes plataformas, ejecutados automática y manualmente, por maquinarias o personas.

El Workflow será, entonces, el encargado de ir ejecutando las diferentes actividades involucradas en un proceso de negocio, invocando diferentes sistemas existentes, otros workflow, etc. En definitiva, el Workflow también oficia de integrador de los diferentes sistemas de la empresa.

Una solución de Workflow no implica una tecnología o producto particular. Existen variadas tecnologías, metodologías y herramientas que permiten construir sistemas automatizados. Para estar seguros de escoger la mejor, se debe analizar cada aplicación en el contexto correspondiente.

Se mencionan a continuación algunas de las características más importantes de los sistemas de workflow, así como las principales funcionalidades que estos deben presentar.

- Se debe entender a la organización como un conjunto de personas que ejecutan y coordinan acciones, en donde cada persona demanda acciones de otra y la calidad de cada acción incide en el valor final. Un Workflow debe traducir el flujo del trabajo en la organización en un conjunto de procedimientos estructurados en donde el énfasis está dado por la coordinación de las acciones a medida que se ejecuta el trabajo.
- Cualquier individuo puede realizar una transacción en la medida en que esté autorizado para hacerlo. Es en ese momento en que la transacción fluye a través de la organización tomando diferentes rutas dependiendo de las acciones que se realizan.
- Un Workflow debe identificar responsables y tiempos de ejecución de cada etapa, proveer información sobre el estado de cada transacción, registrar y desplegar información histórica.
- Se deben soportar situaciones de paralelismo de actividades y de sincronismo de las mismas.
- Intercambio de información con los sistemas de información tradicionales.
- Mecanismos de seguridad.

Un sistema de Workflow proporciona grandes beneficios a las organizaciones que lo utilizan, no únicamente ahorro de tiempo en la administración de la información, sino también muchos otros motivos como ser:

- **Eficiencia en los procesos y estandarización de los mismos:** obteniendo de esta manera un mayor conocimiento de los mismos, lo que nos brinda la posibilidad de realizar mejores controles, obteniendo de esta forma, mejor calidad.
- **Asignación de recursos a tareas:** Las asignación de tareas a recursos puede ser realizada mediante roles, eliminando la tarea de asignar recursos uno a uno, además se pueden desarrollar algoritmos de distribución de recursos a tareas de forma de balancear la carga de cada uno, con la finalidad de maximizar la disponibilidad de los mismos.
- **Diseño de procesos:** Se fomenta a realizar las diferentes tareas de la organización en forma de proceso, documentarlas y en consecuencia, tener un mayor control sobre las mismas.

Para clarificar más los conceptos introducidos, y ver como se aplica un sistema de workflow en un proceso de negocio de una empresa, se dará un breve ejemplo sobre el proceso de solicitud de vacaciones de los empleados de la empresa (Figura 1).

Se detallan a continuación algunas particularidades del proceso:

El empleado que solicita su licencia debe completar un formulario en donde ingresa datos como nombre, fecha de inicio de las vacaciones, fecha de fin de las mismas y su dirección de mail. Una vez ingresado dicho formulario de entrada, debe ser validado por el primer jefe. Éste estudia los datos del empleado y decide si aprueba o no la solicitud, completando un formulario donde ingresa si aprueba o no la solicitud de licencia y el motivo de su decisión. En caso de que el primer jefe no apruebe la solicitud de licencia, se le notifica al empleado vía mail la decisión y se termina el proceso. Si en cambio el primer jefe aprueba la solicitud de licencia, dicha solicitud debe ser validada por un segundo jefe (un gerente general, por ejemplo). Éste estudia la entrada del empleado y la ingresada por el primer jefe, y decide si aprueba o no la solicitud. Para hacerlo debe completar un formulario en donde ingresa si aprueba o no dicha solicitud y los motivos de su

decisión. Una vez que el segundo jefe aprueba o no la solicitud, se pasa al último paso del proceso que es notificar la decisión final vía mail al empleado que hizo la solicitud de licencia.

En la siguiente figura se puede ver de forma gráfica como es la estructura del proceso.

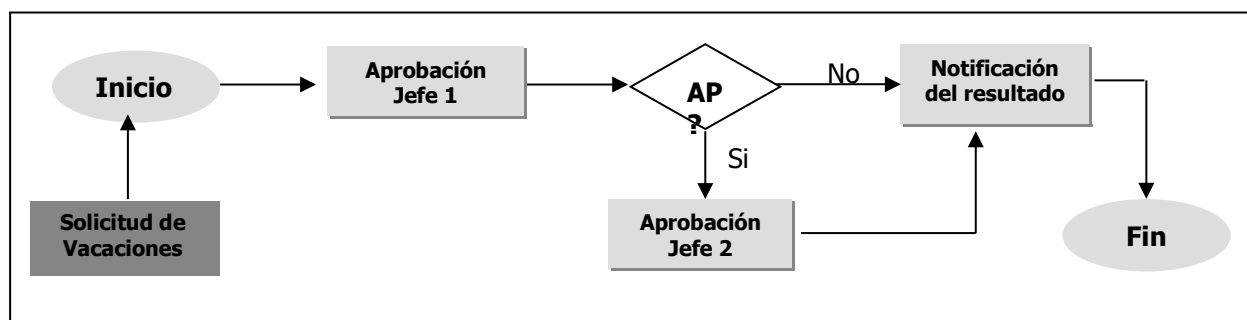


Figura 1

En la antigüedad, este proceso podía realizarse manualmente, manejando documentos y tomando decisiones según sus resultados, lo cual dificulta la administración de los procesos en curso, ya que dichos documentos podían perderse o traspapelarse. Mediante la utilización de sistemas de workflow, este proceso puede automatizarse sencillamente, simplemente definiendo el proceso anterior de la siguiente manera:

- Primero hay que definir como va a ser la información que debe ingresar el empleado que hace la solicitud, se podría definir que haya un formulario en donde el empleado deba completar algunos campos como ser: el nombre, la fecha de inicio de la licencia, la fecha de fin y su dirección de mail. Además se deben definir como es la información que debe completar tanto el primer como el segundo jefe. Para el caso del primer jefe habría que definir que se le muestre en un formulario los datos ingresados por el empleado, y en el cual pueda completar si aprueba o no la solicitud y los motivos de su decisión. Para el caso del segundo jefe habría que definir que en otro formulario se le muestren los datos ingresados tanto por el empleado como por el primer jefe, y en donde pueda completar si aprueba o no la solicitud y los motivos de su decisión.
- Luego se define el proceso, utilizando un editor gráfico que permita diagramarlo fácilmente, y se le asocia la información de entrada al mismo. En este caso, se le debe asociar al proceso el formulario de entrada antes definido para que el empleado haga su solicitud. Con el editor gráfico se van creando las distintas actividades y las transiciones entre las mismas, para obtener así el flujo del proceso. La primer actividad es aquella en la cual el primer jefe aprueba o no la solicitud del empleado. Para esto se crea una actividad llamada "Aprobación Jefe1", y se le asocia un participante de workflow y una estructura de datos. En este caso el participante de workflow responsable de la actividad es el jefe 1 y la estructura de datos es el formulario antes definido para que el jefe vea los datos ingresados por el empleado y pueda ingresar si aprueba o no la solicitud.

En tiempos de ejecución, el proceso va a quedar suspendido en esta actividad hasta que el jefe 1 complete el formulario asociado y recién entonces se continúa con el flujo del proceso. La próxima actividad a insertar es una actividad de decisión en la cual se evalúa una determinada condición y según el resultado de esa evaluación el flujo del proceso va hacia una actividad u otra. La condición a evaluar es si el primer jefe aprobó las vacaciones, en caso afirmativo el flujo va hacia una actividad llamada "Aprobación Jefe2" en donde un segundo jefe debe aprobar la solicitud, en caso de que la evaluación sea negativa el flujo va hacia la actividad "Notificación del resultado".

Luego se define la actividad llamada "Aprobación Jefe2" a la cual se llega solo si el primer jefe aprobó la solicitud del empleado. De igual forma que para la actividad "Aprobación Jefe 1", a esta actividad se le asocia un participante de workflow responsable que va a ser el que apruebe o no la solicitud del empleado, y una estructura de datos que son los datos que debe completar el responsable de la actividad. En este caso el participante responsable será el segundo jefe, y la estructura de datos que debe completar es un formulario en donde puede ver la información ingresada tanto por el empleado como por el segundo jefe y en donde debe completar si aprueba o no la solicitud y los motivos de su decisión. Igual que para el caso anterior, cuando el proceso llega

a esta actividad se suspende esperando que el segundo jefe complete el formulario asociado a esta actividad.

Por último, se deberá ingresar una actividad en la cual se le comunica al empleado si su solicitud fue aceptada o no. El nombre de esta actividad es "Notificación del resultado", y puede ser alcanzada tanto desde la actividad "Aprobación Jefe 1", si el primer jefe reprueba la solicitud, como de la actividad "Aprobación Jefe 2".

Esta actividad debe mandar un mail al empleado que hizo la solicitud, enviado a la dirección ingresada por el empleado en el formulario de entrada al proceso y en el cuerpo del mail se le comunica cual fue el resultado final de su solicitud.

El proceso antes definido, es ejecutado por el sistema de workflow de la siguiente forma:

- En el momento en que se inicia el proceso, se despliega el formulario de entrada al mismo en donde el empleado debe ingresar sus datos (nombre, fecha de inicio, fecha de fin y dirección de mail para poder ser contactado).
- Luego que el empleado haya completado el formulario de entrada con sus datos, el flujo del proceso pasa a la siguiente actividad, que es la actividad llamada "Aprobación Jefe1". Cuando el proceso llega a esta actividad, se suspende y queda esperando que el primer jefe complete el formulario asociado (donde puede ver los datos ingresados por el empleado) y debe ingresar si aprueba o no la solicitud y cual es el motivo de su decisión. En el momento en que el primer jefe complete este formulario, se pasa a la siguiente actividad del proceso.
- Una vez que el primer jefe completó el formulario asociado a la actividad "Aprobación Jefe1", se pasa a la actividad de decisión, donde es evaluada la decisión del primer jefe. En caso de que se haya aprobado la solicitud se pasa a la actividad llamada "Aprobación Jefe2". En cambio, si el primer jefe rechazó la solicitud se pasa a la actividad "Notificación del resultado".
- En caso de que el primer jefe haya aprobado la solicitud se pasa a la actividad "Aprobación Jefe2", entonces el proceso queda suspendido esperando que el segundo jefe complete el formulario asociado a esta actividad. En dicho formulario, el segundo jefe puede ver toda la información del empleado y el primer jefe. Debe ingresar si aprueba o no la solicitud. Una vez que el segundo jefe completa el formulario se pasa a la actividad "Notificación del resultado".
- La última actividad del proceso es la actividad "Notificación del resultado" que manda un mail al empleado, con el resultado de su pedido. Luego de esta actividad el proceso queda completado.

Como se puede observar, una vez iniciado el proceso, el empleado ingresa la información del formulario de entrada y las restantes actividades son ejecutadas automáticamente por el sistema de workflow, exceptuando aquellas que son realizadas por usuarios del sistema. De esta forma, se obtiene mayor control y agilidad en la ejecución del proceso, asegurándose que los documentos sigan la ruta deseadas, sin la necesidad de manejar expedientes con las problemáticas que estos conllevan. Una ventaja adicional, es que cada participante del proceso, en cada momento, puede consultar sus tareas pendientes sin necesidad de hacer una búsqueda en documentos.

Procesos de este estilo abundan en las organizaciones de hoy en día. Por este motivo, un sistema de workflow es de vital importancia. Las empresas no siempre pretenden un sistema de workflow completo. Es probable que simplemente quieran automatizar un proceso de negocio específico. Por tanto, un sistema de workflow puede convertirse en una plataforma de desarrollo de sistemas, ya que por ejemplo, si una empresa necesitara un sistema como el descrito anteriormente, alcanzaría con realizar el proceso como se ha visto, brindarle una interfaz amigable y desarrollar aplicaciones que asistan al proceso, para tener pronto el sistema en poco tiempo.

1.3 Descripción del proyecto.

En este proyecto, se propone realizar un estudio profundo sobre los sistemas de workflow y la posterior implementación de un prototipo. Se espera obtener un sistema que permita definir procesos de negocios de forma amigable, ejecutarlos y administrarlos. Para definir un proceso de negocio, se deberá utilizar una herramienta gráfica, que represente el flujo como una red de actividades con sus relaciones, criterios para indicar el comienzo o fin del proceso e información sobre las actividades individuales. Es importante también definir el tipo de información que necesita el proceso para su ejecución, validar dicha información y persistirla. Para poder realizar el proyecto, se estudió e investigó los sistemas de workflow, utilizando diversa bibliografía referente a dichos sistemas, estudiando la estructura genérica de los sistemas de workflow, sus principales componentes, entidades, atributos, y las diferentes interfaces que estos deben presentar [1], [4]. Dicho estudio y los requerimientos en cuanto a plataforma y lenguaje nos obligo a introducirnos en temas como ser .Net [20], C# [21], Web Services [22], Microsoft Message Queue [23], Remoting [24], sistemas cliente/servidor [2], XML-XSL [25], XPD [5], [6], [7], [8].

El resultado obtenido es un sistema de workflow con las siguientes funcionalidades:

- Herramienta de diseño de procesos.
- Motor de ejecución.
- Editor automático de formularios / Esquemas.
- Manejador genérico de participantes.
- Herramientas auxiliares para crear flujos.

Las funcionalidades anteriormente descritas, son detalladas con mayor profundidad en el capítulo referente al alcance del proyecto, donde se describe cada una de ellas.

Podemos concluir que la realización del proyecto aportó valiosa información acerca de la estructura general y del funcionamiento de un sistema de workflow, obteniendo un prototipo de gran utilidad para desarrollos futuros.

1.4 Contexto del Proyecto

El proyecto se realizó para la empresa Infocorp, bajo la tutoría de Guillermo Calderón, mientras que por parte de la Empresa, el usuario responsable fue Daniel Gómez.

En la actualidad Infocorp cuenta con un conjunto de soluciones enfocadas a resolver la dinámica y procesos de negocios de una empresa de mediano porte utilizando sistemas desarrollados en su gran mayoría utilizando tecnologías Web con un concepto de "llave en mano".

El proyecto requiere la implementación de una solución de WorkFlow que pueda constituirse como un producto independiente, así como también que sirva como base para todos los productos (implementaciones prácticas del WorkFlow) que la empresa ofrece.

Desde el inicio del proyecto, se trabajó directamente en la empresa, en donde cada uno de los estudiantes contaba con una computadora, con todo el software instalado para el desarrollo del prototipo del workflow, así como acceso a Internet para la búsqueda de material.

Los estudiantes concurríamos a la empresa de lunes a viernes en un promedio de 4 horas por día. En la empresa se nos brindó material y cursos de apoyo que fueron de gran ayuda para comprender rápidamente la plataforma y lenguaje a utilizar.

Fue un requisito de la empresa, que durante la etapa de investigación se generaran documentos con formato de resumen ejecutivo, que describan cada uno de los temas estudiados. Además de los documentos se tenían reuniones periódicamente con el encargado del proyecto por parte de la empresa, en donde se discutían y transmitían los conceptos aprendidos por cada uno de los estudiantes.

Durante la etapa de desarrollo, al principio de cada semana se elaboraba un plan de trabajo que describía lo que iba a realizar cada estudiante durante esa semana, ese plan de trabajo estaba dentro del contexto de un plan global.

Al estar trabajando directamente en la empresa se tenía contacto permanente con el responsable, se hacían reuniones donde se mostraba el avance por parte de los estudiantes y en donde se discutían los resultados obtenidos.

También se mantuvieron reuniones con el tutor por parte de la Facultad, en donde se comunicaba lo que los estudiantes estaban realizando en la empresa y el avance del proyecto.

1.5 Síntesis del informe

A continuación se presentará cada uno de los capítulos que conforman este documento, de forma de facilitar la lectura del mismo.

En el primer capítulo se presenta levemente el proyecto realizado a modo de guiar al lector sobre el contenido del documento, haciendo una breve reseña histórica sobre la evolución de los sistemas de workflow, presentando el contexto en la que se realizó el proyecto, una leve introducción a los sistemas de workflow, brindado su definición, principales ventajas y una primera presentación del problema.

En el segundo capítulo se describe más profundamente los sistemas de workflow, brindado un marco teórico que permita comprenderlos más. Se presenta el prototipo a realizar, mostrando los componentes principales de un sistema de workflow genérico, sus interfaces, y diferentes propiedades.

Una vez analizados los sistemas de workflow, se define, en el tercer capítulo, el alcance del proyecto, definiendo que componentes y herramientas se desarrollarán en el prototipo a implementar, presentando también un cronograma tentativo y análisis de los posibles riesgos que el proyecto puede sufrir.

En el quinto capítulo se presenta la solución realizada, detallando la plataforma y lenguaje utilizados, la arquitectura seleccionada, las herramientas desarrolladas, principales componentes, diagramas de paquetes y deploy, etc. En el capítulo siguiente se presenta brevemente la metodología de testeo utilizada.

Por último se presenta un capítulo que describe los trabajos a futuro a realizar, que continuarán el estudio y prototipo implementado, y un último capítulo en el que se señalan las principales conclusiones obtenidas como resultado del proyecto.

Cada uno de los temas tratados en este documento se presenta de forma más detallada en diferentes apéndices, para aquellos lectores que pretendan profundizar en cada uno de los diferentes temas, obteniendo información más completa y elaborada.

2 DESCRIPCIÓN DEL PROBLEMA.

La finalidad del proyecto, es la construcción de un prototipo de una herramienta que permita crear y automatizar procesos de negocio. En la figura 1, se indican los principales conceptos relacionados con los workflow y sus respectivas relaciones [1].

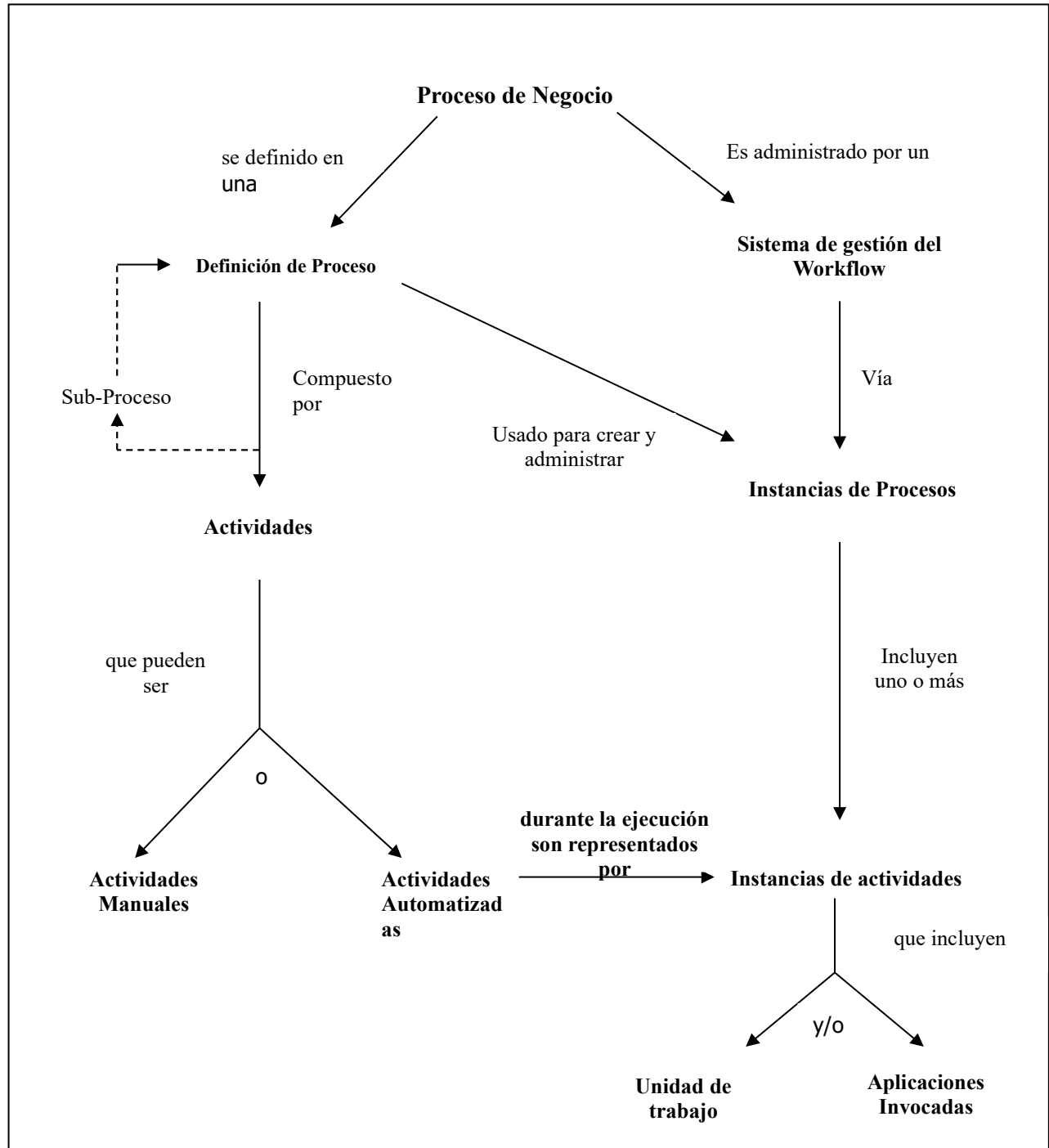


Figura 1

A continuación se definen cada uno de los conceptos presentados en la figura 1.

Proceso de Negocio: Es un conjunto de uno o más procedimientos o actividades vinculadas, las cuales cubren colectivamente un objetivo del negocio, normalmente dentro del contexto de una estructura organizacional que define roles funcionales y sus relaciones.

Definición de Proceso: Es la representación de un proceso de negocio, de manera que permita la manipulación automatizada, como ser su modelado y ejecución por un sistema de gestión de workflow. La definición de un proceso consiste en una red de actividades y sus relaciones, criterios para indicar el inicio y fin e información acerca de las actividades individuales, como ser participantes, aplicaciones asociadas e información particular del mismo.

Sistema de Gestión de Workflow: Es un sistema que define, crea y administra la ejecución de workflows a través del uso de software, corriendo sobre uno o más motores de workflow, capaces de interpretar la definición de proceso, interactuar con participantes de workflow y, donde es requerido, invocar el uso de herramientas y aplicaciones externas.

Actividad: Es la descripción de una pieza de trabajo que representa un paso lógico en un proceso. Una actividad puede ser manual o automatizada. La misma requiere recursos humanos o computarizados para la ejecución de un proceso.

Actividad Automatizada: Es una actividad que es capaz de automatización computarizada usando un sistema de gestión de workflow para administrar la actividad durante la ejecución de un proceso de negocio del cual forma parte.

Actividad Manual: Es una actividad en un proceso de negocio la cual no es capaz de automatización, por lo cual, no forma parte del alcance de un sistema de gestión de workflow. Estas actividades deben ser incluidas en la definición de procesos.

Instancias de Actividad o Proceso: Es la representación de la ejecución de un solo proceso o actividad, incluyendo sus datos asociados. Cada instancia representa un hilo independiente de ejecución de un proceso o actividad, el cual puede ser controlado independientemente, teniendo su propio estado interno e identidad externamente visible, la cual es usada para almacenar u obtener información relacionada con cada ejecución individual.

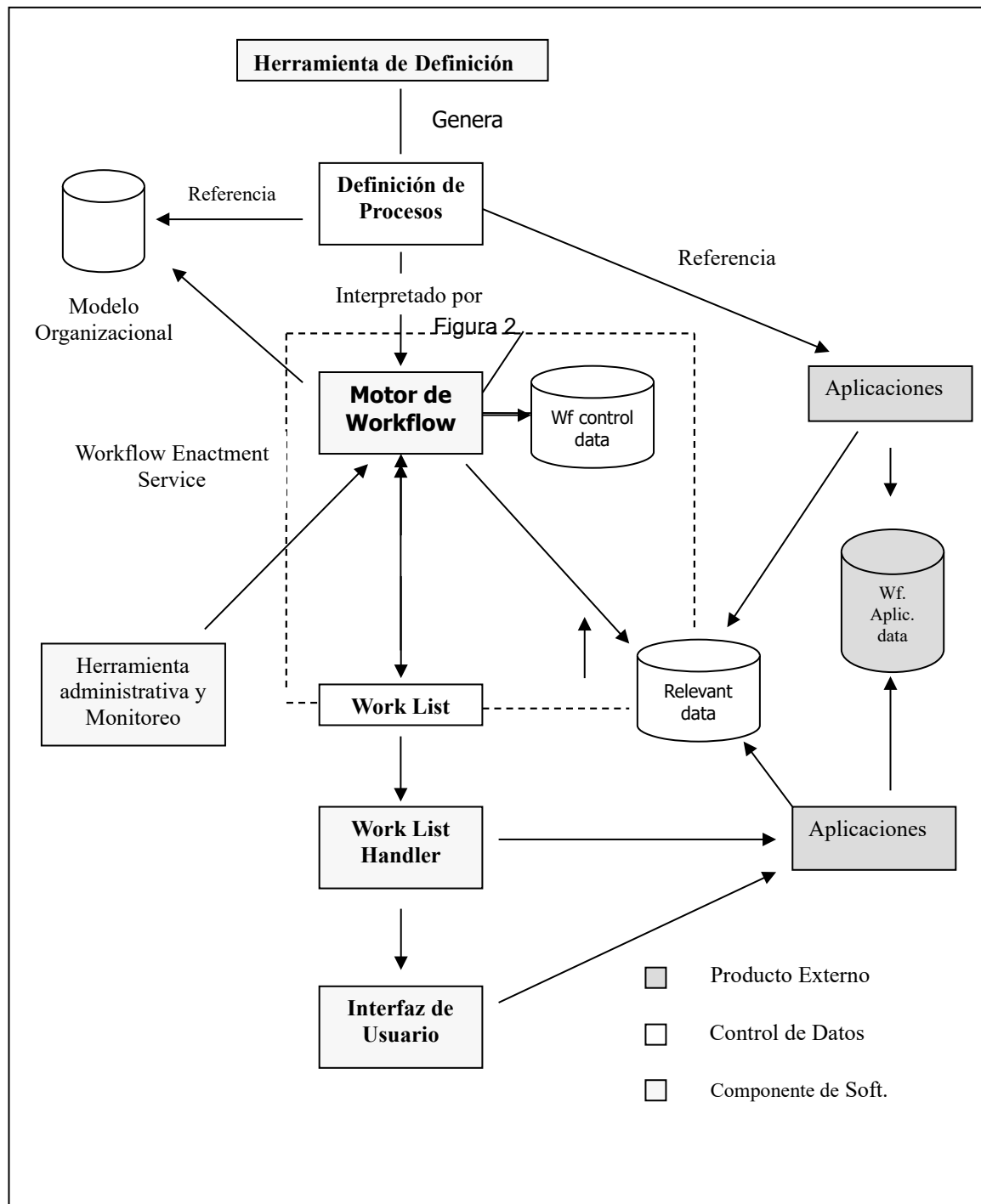
Unidad de Trabajo: (WorkItem) Es la representación del trabajo a ser procesado (por un participante de workflow) en el contexto de una actividad en una instancia de proceso. Es importante tener en cuenta que una de las principales características de un sistema de workflow es la capacidad de interactuar con usuarios y la asignación de tareas a los mismos.

Aplicación Invocada: Aplicaciones usadas para permitir realizar una determinada actividad.

Una vez definidos los conceptos básicos relacionados a los workflows, estamos en condiciones de especificar el problema de forma más detallada, describiendo la estructura general de un sistema de gestión de workflow, sus componentes e interfaces más comunes. Esto será de gran ayuda y un buen punto de partida para resolver el problema [3], [4].

2.1 Estructura Genérica de un Producto de Workflow.

En un producto de software de Workflow genérico, se identifican una serie de componentes e interfaces. La implementación de esta estructura puede ser realizada de varias formas diferentes, siendo éste un punto de desencuentro de los productos existentes [13], [14], [15].



En este modelo genérico presentado en la Figura 2, encontramos tres tipos de componentes:

- De software:
Proveen soporte para gran cantidad de funciones del sistema de Workflow.
- Datos y Definición de procesos: Usados por los componentes de software.
- Aplicaciones externas.

A continuación se describirán los componentes más importantes:

2.1.1 Herramienta de Definición de Procesos.

Forma parte de las componentes de software del Workflow y se la puede ver en el borde superior de la figura. Es utilizada para crear una descripción de los procesos en una forma procesable para una computadora. Podría estar basada en un lenguaje de definición de procesos formal, en un modelo de interacción entre objetos, o simplemente en un conjunto de reglas de ruteo para transferir información entre los participantes.

Esta herramienta puede ser proporcionada como parte de un producto de software orientado a Workflow, o podría simplemente existir por si sola y tener integración con diferentes productos de Workflow [5]. [6], [7], [8].

2.1.2 Definición de Procesos.

Luego de la componente anterior, encontramos la Definición de Procesos, que forma parte de los datos del Workflow. Contiene toda la información necesaria acerca de los procesos, incluyendo información de comienzo de actividades, condiciones, y reglas de navegación.

Podría tener referencias a la definición de roles, donde se almacena información de la estructura organizacional. Esto quiere decir que en la definición de procesos se puede mencionar que en cierto proceso participa cierto rol, el cual esta definido en la definición de roles.

2.1.3 Workflow Enactment Service.

El WES interpreta la descripción de procesos y controla las diferentes instancias de los mismos, la secuencia de actividades, adiciona ítems (elementos) a la lista de trabajo de los usuarios (Worklist), e invoca aplicaciones necesarias. Todas estas tareas son hechas por uno o más motores de Workflow (engines), los cuales manejan la ejecución de las distintas instancias de varios procesos. Mantiene también datos de control interno, tanto centralizado como distribuido por un conjunto de motores de workflow; estos datos de control del workflow incluyen la información de estado interno asociada a las varias instancias de procesos e instancias de actividades bajo ejecución. El motor de workflow debe incluir alguna capacidad de invocación a aplicaciones, para poder invocar las aplicaciones necesarias para ejecutar actividades particulares. En general, los mecanismos con este fin pueden variar muchísimo. Algunos sistemas simples, pueden solo ofrecer una única herramienta, como un editor de texto o formularios, mientras otros pueden ofrecer métodos para la invocación de un amplio rango de herramientas, tanto locales como remotas al motor de workflow.

2.1.4 Datos relevantes y datos de aplicaciones

La toma de decisiones de navegación u otras operaciones de control en el motor de workflow, se basa en datos generados o actualizados por aplicaciones de workflows. Dichos datos son accesibles al motor de workflow, y llamados "workflow relevant data", siendo el único tipo de datos de aplicación que es accesible por el motor de workflow. La información de aplicaciones, es manipulada directa y solamente por las aplicaciones invocadas, además los motores de workflow pueden ser responsables por transferir dichos datos entre aplicaciones (si es necesario).

2.1.5 Worklist (lista de trabajo)

La Worklist forma parte de los datos del Workflow y la podemos apreciar en la parte inferior de la figura. Dado que la interacción con los usuarios es necesaria en algunos casos, el motor de Workflow utiliza una WorkList, manejada por un manejador de WorkList, para controlar tal interacción. El motor deposita en la WorkList ítems ha ser ejecutados para cada usuario. La Worklist puede ser visible o invisible para los usuarios. Dependiendo del caso, muchas veces se deja que el usuario seleccione ítems y los procese en forma individual.

2.1.6 Manejador de Worklist.

Luego del componente anterior, se encuentra el Manejador de la Worklist. Es un componente de software que maneja la interacción entre los participantes del Workflow y el Workflow Enactment Service (WES), vía la Worklist. El manejador soporta en general un amplio rango de interacción con otras aplicaciones clientes. En la figura, la interfase con el usuario es mostrada como una componente separada del manejador de Worklist. En algunos sistemas estas dos componentes están agrupadas como una única entidad funcional.

2.1.7 Operaciones de supervisión

En un sistema de workflow hay numerosas funciones de supervisión que se proveen normalmente. Éstas, son típicamente brindadas sobre privilegios de supervisión a una estación de trabajo o usuario.

Estas funciones pueden permitir a los supervisores alterar reglas de asignación de trabajo, identificar participantes para roles organizacionales específicos en un proceso, rastrear alertas para fechas límites, rastrear el historial de una instancia de proceso particular, etc.

Como se puede visualizar en la Figura 2, el Workflow está compuesto de diferentes módulos. Surgen, entonces, diferentes posibilidades a la hora de diseñar un sistema de Workflow. En primera instancia, se puede ver al Workflow como un conjunto de herramientas (motor, herramientas administrativas, herramientas de definición de procesos, etc.). Sin embargo, la idea de considerar al Workflow independientemente de las demás herramientas parece más atractiva. De esta forma, es posible adquirir las herramientas consideradas óptimas, sin importar que estas sean desarrolladas por diferentes empresas, siempre que respeten un estándar. Esta política brinda también la opción de definir interfaces de forma clara, permitiendo modificar o realizar nuevamente cada uno de los componentes, sin que esto impacte en el resto del sistema.

En la Figura 3, se indican las interfaces que define la WFMC y se puede ver que coinciden con la estructura genérica del workflow anteriormente descrita, y con el sistema a realizar.

No es de interés en el proyecto, realizar las interfaces como se indican en la WfMC, pero si definirlas de forma clara. Las Interfaces 1, 2, 3 y 5 son de particular interés ya que son fundamentales para el funcionamiento del sistema y es posible que la modificación de estos componentes suceda con frecuencia, no así la Interfase 4, la cual permite la interoperabilidad con otros sistemas de Workflow.

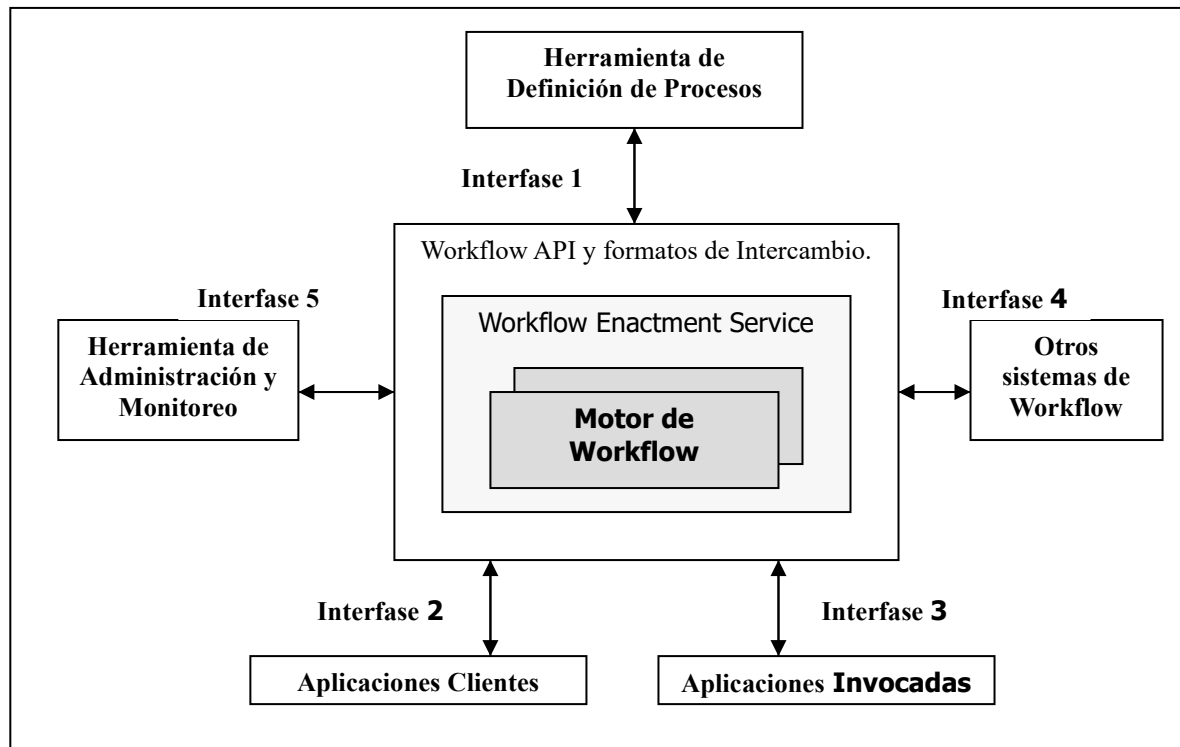


Figura 3

Analizando esta estructura general de un sistema de workflow, se concluye que se realizará la totalidad de las herramientas anteriormente señaladas, ya que es necesario realizar un prototipo totalmente funcional, lo que implica: la posibilidad de diseñar procesos, ejecutarlos, interactuar con usuarios, monitorear el sistema, además se especificaran las interfaces correspondientes a cada uno de los componentes, exceptuando la interfaz 4. Al decir un prototipo totalmente funcional, queremos decir que todas estas funcionalidades son indispensables, pero no tienen porque estar completamente realizadas cada una de ellas. [8], [9]. [10], [11], [12].

3 ALCANCE DEL PROYECTO

El objetivo del proyecto es implementar una solución de workflow, que sea, además, una plataforma tecnológica que permita actualizar los productos de la empresa y desarrollar un marco de trabajo unificado, el cual posibilite optimizar tiempos de desarrollo tanto de productos paquete como de desarrollo a medida.

3.1 Funcionalidades a Desarrollar.

En esta sección se describirán las principales funcionalidades que debe proveer el prototipo a realizar.

3.1.1 Editor Visual de procesos.

Se deberá realizar una herramienta que permita realizar flujos de trabajo. La misma debe ser amigable para el usuario, analizando la posibilidad de que pueda accederse desde la web.

El editor debe proveer la capacidad de crear y modificar procesos de negocios simples, los mismos constan de:

- Nodos de Inicio y Fin (deben ser únicos por proceso).
- Nodos que representen una actividad a ejecutarse por el workflow.
- Nodos que representen una decisión a tomar por el sistema según la ejecución del flujo.
- Nodos que representen el estado en que se encuentra el flujo (hitos)
- Aristas que representen el flujo que debe seguir el sistema.

Se debe proveer un lenguaje que permita, a las actividades, acceder a los datos que se van generando a medida que se ejecuta el mismo.

El editor deberá almacenar los procesos de forma que el motor del workflow pueda interpretarlos y ejecutarlos.

3.1.2 Editor de Esquemas/Formularios.

A la hora de definir como se realizaría la administración de la información en el WorkFlow, pretendimos buscar una forma de hacerlo que fuese suficientemente libre como para no acotar las posibilidades de uso, y suficientemente estructurada para ser fácilmente administrable por nosotros, para no tener que realizar, por ejemplo, interfaces particulares para cada etapa de un proceso, o herramientas personalizadas para cada aplicación que impacten los cambios en base de datos. A la vez, la administración debería ser adecuada para permitir, en pasos futuros, si bien no es la idea en la primera instancia, integrar nuestro WorkFlow con otras herramientas similares de forma relativamente sencilla. Para lograrlo, debemos tener una forma de administración de la información medianamente estructurada. Por todos estos motivos, creemos que la mejor forma, es trabajar con archivos XML, para permitir una interacción transparente, a la vez que un tratamiento genérico. Intentamos darle una potencialidad mayor, como para que a partir de un esquema se pueda generar de forma genérica las páginas web necesarias para ingreso y visualización de datos, y el impacto automático de éstos en base de datos. Yendo un poco más allá, pretendemos también que nuestra estructura de almacenamiento de la información, permita cierta transparencia con respecto a la fuente física de almacenamiento, por lo tanto debe permitir trabajar desconectado de la fuente y actualizarse con esta, cuando se evalúe conveniente.

Por todo lo expuesto anteriormente, se definió el concepto de "esquema", que permite definir la estructura relacional, junto a una serie de propiedades extra, asociadas a tablas, columnas o relaciones. Se deben definir entonces, tres herramientas. Una que permita definir un esquema de la información que deberá viajar entre las distintas actividades de un proceso; una que, a partir del esquema, genere el formulario para presentar la información en pantalla; una que, tomando una instancia de un esquema, impacte los cambios en la base de datos, desde la última actualización (desde la cual trabaja desconectado).

A tales efectos el esquema debe permitir, no solo definir su estructura relacional (tablas, columnas, relaciones, etc.) si no también una serie de propiedades extra, necesaria para soportar la actividad de la edición genérica de formularios (por ejemplo, descripciones de columnas y tablas, visible o no, modificable o no, scripts de validación de datos ingresados, rangos de validez, etc.) y su impacto automático en base de datos (nombre físico de campos y tablas, si acepta o no valores nulos, etc.). Luego, las otras dos herramientas deben utilizarlos de la forma adecuada.

3.1.3 Motor de Ejecución.

Se debe realizar un motor que permita interpretar el flujo generado con el editor. El mismo deberá poder ejecutar procesos simples, invocar las aplicaciones correspondientes, modificar el estado del proceso y tomar decisiones según los resultados de las actividades. Además de posibilitar pausar, reiniciar y detener la ejecución de un proceso.

También debe permitir ejecutar actividades que dependan de la confirmación de usuarios (mediante el uso de Listas de trabajo o WorkList), o ingreso de información de los mismos, en este caso el motor debe terminar la ejecución del proceso, ya que esta tarea podría demorar un tiempo indeterminado, y continuarla cuando el usuario ingrese la información correspondiente.

3.1.4 Gestor genérico de participantes.

Deberá desarrollarse un componente que permita manejar participantes, esto es, crear, modificar y eliminar los posibles participantes del sistema de workflow. El mismo deber permitir acceder a diferentes repositorios, de forma que sea transparente para el sistema. Dichos participantes pueden ser asignados a diferentes tareas, por este motivo, debe realizarse una interfase que permita seleccionar los participantes a asignar sin importar su fuente.

3.1.5 Herramientas auxiliares para ejecutar flujos.

Se deberá realizar una batería de herramientas que puedan referenciarse desde el editor de flujos, que realicen diferentes actividades. A continuación se enumeran las mismas.

- **WebServices:** Permite invocar webservices, se debe ingresar la url correspondiente al webservice, el nombre del método a invocar, los parámetros de entrada, y se devuelve un XML con el resultado de la invocación a dicho webservice
- **Scripts:** Permite la ejecución de scripts ya sea en los lenguajes JScript o VBscript, se debe seleccionar el lenguaje de script e ingresar el código del mismo.
- **Assemblies:** Permite la invocación de métodos que se encuentran en una determinada clase compilados en assemblies (dll's), dada la dirección del assembly, el nombre de la clase, el método a invocar y los parámetros de entrada se realiza la invocación correspondiente.
- **Mails:** Permite el envío de mails, se deben ingresar los datos que comúnmente se ingresan al enviar cualquier mail como ser, destinatario del mail, quienes reciben copia, quienes reciben copia oculta, cual es el asunto del mail, el cuerpo del mail, también se deberá ingresar el servidor de mail saliente (SMTP).

4 PLANIFICACIÓN DEL PROYECTO

4.1 Cronograma.

Se presentará a continuación el cronograma tentativo del proyecto, señalando las diferentes tareas. El mismo se extiende desde el 15/5/2003 hasta el 31/03/2004, constando de cuatro etapas que se indican a continuación:

Nº	Etapas	Duración estimada
1	Estudio de Sistemas de Workflow	1 mes
2	Editor de flujos Web (Análisis, Desarrollo, Testeo) Editor de esquemas (Análisis, Desarrollo, Testeo) Generador de formulario (Análisis, Desarrollo, Testeo)	3 meses
3	Motor (Análisis, Desarrollo, Testeo) Gestor de Participantes (Análisis, Desarrollo, Testeo) "ToolAgents"	4 meses
4	Documentación Presentación	2 meses

A continuación se presenta la lista de supuestos sobre la cual se apoya esta propuesta:

- Disponibilidad a materiales de estudio y/o recursos adicionales que se crean convenientes
- Capacidad de compra para productos de terceros, facilitadores a la hora de resolver problemas técnicos
- Infraestructura adecuada según los requerimientos de las tecnologías y herramientas involucradas
- Canales de comunicación rápidos para mantener al equipo unido e informado.

4.2 Riesgos

Los riesgos de proyecto son características, circunstancias o elementos del ambiente de proyecto que pueden tener un efecto adverso en la calidad del proyecto o de los entregables producidos. Los riesgos identificados se incluyen en la siguiente lista. Se aplicará un plan para minimizar o eliminar el impacto de cada riesgo en el proyecto.

Riesgo	Mitigar
Interferencia de proyectos "facturables" que afecten a los técnicos designados para este proyecto	En la sección de supuestos se detalla la asignación de horas requeridas para cada uno de los recursos.
Desviación de fechas estimadas de proyecto	Reunión de avance semanal. Reporte semanal de horas insumidas.

5 SOLUCIÓN AL PROBLEMA.

5.1 Lenguaje y plataforma utilizados

La plataforma utilizada fue .NET y el lenguaje C#. Tanto la plataforma como el lenguaje fueron requerimientos de la empresa en donde se realizó el proyecto, aparte de ser interesantes y con proyección de futuro, por ser herramientas nuevas y que, aparentemente, cuentan con todo el respaldo y el empeño de Microsoft, como para que se implanten en el mercado [19].

.NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. Ésta es la llamada plataforma .NET, y a los servicios antes comentados se les denomina servicios Web.

C# es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET

La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic. Ver Apéndice - Introducción a C# (Estudios Técnicos)

A continuación, se resumen las ventajas más importantes que proporciona .Net Framework:

Código administrado: El CLR realiza un control automático del código para que este sea seguro, es decir, controla los recursos del sistema para que la aplicación se ejecute correctamente.

Interoperabilidad multilenguaje: El código puede ser escrito en cualquier lenguaje compatible con .Net ya que siempre se compila en código intermedio (MSIL).

Compilación just-in-time: El compilador JIT incluido en el Framework compila el código intermedio (MSIL) generando el código máquina propio de la plataforma. Se aumenta así el rendimiento de la aplicación al ser específico para cada plataforma.

Garbage collector: El CLR proporciona un sistema automático de administración de memoria denominado recolector de basura (garbage collector). El CLR detecta cuándo el programa deja de utilizar la memoria y la libera automáticamente. De esta forma el programador no tiene por que liberar la memoria de forma explícita aunque también sea posible hacerlo manualmente (mediante el método dispose()) liberamos el objeto para que el recolector de basura lo elimine de memoria).

Seguridad de acceso al código: Se puede especificar que una pieza de código tenga permisos de lectura de archivos pero no de escritura. Es posible aplicar distintos niveles de seguridad al código, de forma que se puede ejecutar código procedente del Web sin tener que preocuparse si esto va a estropear el sistema.

Despliegue: Por medio de los ensamblados resulta mucho más fácil el desarrollo de aplicaciones distribuidas y el mantenimiento de las mismas. El Framework realiza esta tarea de forma automática mejorando el rendimiento y asegurando el funcionamiento correcto de todas las aplicaciones.

Se enumerarán a continuación algunas de las desventajas de .Net Framework:

Procesos como la recolección de basura de .Net o la administración de código introducen factores de sobrecarga que repercuten en la demanda de más requisitos del sistema.

El código administrado proporciona una mayor velocidad de desarrollo y mayor seguridad de que el código sea bueno. En contrapartida el consumo de recursos durante la ejecución es mucho mayor, aunque con los procesadores actuales esto cada vez es menos inconveniente.

El nivel de administración del código dependerá en gran medida del lenguaje que utilicemos para programar. Por ejemplo, mientras que Visual Basic .Net es un lenguaje totalmente administrado, C Sharp permite la administración de código de forma manual, siendo por defecto también un lenguaje administrado. Mientras que C++ es un lenguaje no administrado en el que se tiene un control mucho mayor del uso de la memoria que hace la aplicación.

C# es un lenguaje simple y moderno, orientado a objetos que puede ser usado inmediatamente por programadores de C y C++.

Elimina costosos errores de programación. Por ejemplo, hasta un programador experto de C++ puede cometer el mas simple de los errores como ser, no inicializar una variable, esta clase de errores pueden resultar en problemas impredecibles, que pueden quedar sin descubrir por largo períodos de tiempo. En C# las variables se inicializan automáticamente.

Más ventajas:

El garbage collector Libera al programador del peso de administrar manualmente la memoria.

Variables de tipo seguro ("type-safe").

También cuenta con la posibilidad de actualizar los componentes de software, ya que C# incluye soporte para el "versionado" (versioning support) dentro del lenguaje, el cual no se puede hacer ni en C++ ni en Java. Esto ayuda a prevenir errores de código y preservar la flexibilidad de versionado.

Las ventajas de C# sobre VB.NET son que la sintaxis de C# es un poco mas clara que la de VB, y que C# permite insertar bloques ("inseguros") de código C++ puro en situaciones donde se necesita el poder de los punteros.

Hay similitudes sintácticas entre el código escrito en C# y el código escrito en Java. Esto se debe, según Microsoft a que tanto C# como Java son modernizaciones de C++. Sin embargo C# conserva mas los poderes de C++ que Java.

5.2 Arquitectura

La arquitectura ayuda a identificar los elementos estructurales que pueden ser útiles en la construcción de sistemas. Permite determinar la estructura y forma de los sistemas que podemos construir para la satisfacción de diversas necesidades.

A la hora de diseñar el sistema de Workflow, se pensó fundamentalmente en una arquitectura que sea altamente escalable y de fácil distribución. Es por este motivo que se optó por la arquitectura que se muestra en la siguiente figura:

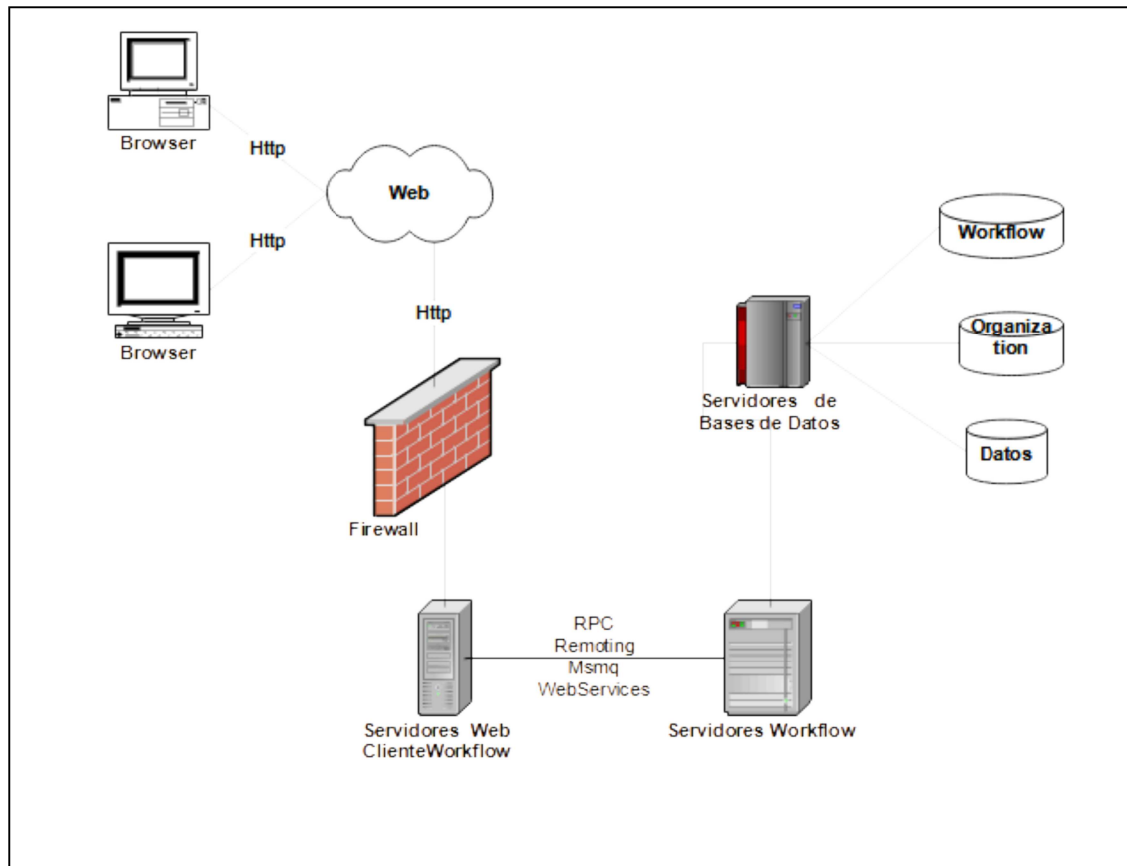


Figura 4

Para hacer que el producto sea de sencilla distribución, tanto las herramientas administrativas, de monitoreo, como las diferentes aplicaciones que utilizan el Workflow son aplicaciones Web. De esta forma, al instalar una nueva versión del producto, simplemente hay que realizar la modificación en el servidor Web donde estén las mismas sin tener impacto alguno para el usuario (la distribución web es muy simple). Es importante resaltar que esta opción no pierde escalabilidad ya que se pueden instalar tantos servidores Web como sea necesario, según el uso que se le de al sistema. De forma simple y, mediante administración de Windows, se pueden agregar tantos servidores web como se considere necesario, desarrollando una granja de servidores. Esto es mera administración de Windows 2000 Server, sin tener impacto alguno en las aplicaciones en ellos instaladas. Si bien las diferentes aplicaciones fueron implementadas de forma Web, se dejó la puerta abierta para desarrollarlas de cualquier forma, utilizando una interfaz bien definida.

Desde el punto de vista de la escalabilidad, se ha evaluado que muy probablemente, si bien es importante poseer la habilidad de utilizar sencillamente varios servidores Web, el problema no está en ellos, sino que el que se verá más exigido será el servidor que posea el motor del Workflow. Por tanto, se ha considerado especialmente la posible separación de las aplicaciones Web y del motor mismo. Se ha considerado

también, utilizar uno de los mecanismos descritos en la figura, para la comunicación entre las aplicaciones y el motor (Msmq, Remoting, etc.) De esta forma, se puede tener varios servidores ejecutando el motor de Workflow, tomando los pedidos realizados a través de la red (por medio de cualquiera de los mecanismos mostrados).

Además cada motor de Workflow estará utilizando servidores de base de datos, los cuales estarán configurados como sea conveniente según la carga del sistema. De esta forma atacamos todas las posibles debilidades el producto, ya sea por excesivo uso de las páginas Web, utilizando varios servidores Web, también ofrecemos una solución al problema de un intenso uso del motor de Workflow, pudiendo este ser instalado en varios servidores, y también la base de datos puede ser sencillamente potenciada.

Se puede definir la arquitectura de nuestro sistema como una arquitectura cliente/servidor, ya que, como se deduce del termino, clientes y servidores son entidades lógicas independientes que operan en conjunto a través de una red para realizar una tarea [2].

En un sistema cliente / servidor existen tres componentes principales que se detallaran a continuación:

Cliente: Ejecuta la parte del Cliente de la aplicación. En general, ofrece una interfaz grafica que permite al usuario acceder a los servicios distribuidos, dondequiera que se encuentren. (Consumidor de recursos, proactivo)

Servidor: Ejecuta la parte del Servidor de la aplicación. Posee los servicios que se quieran ejecutar, e interactúa con otros sistemas como ser DBMS. (Proveedor de recursos, reactivo)

Middleware: Corre tanto en el cliente como en el servidor. Este es el sistema nervioso del sistema Cliente/Servidor y es el enlace que permite que un cliente obtenga un servicio de un servidor. Este empieza en el modulo que presenta las API's al cliente para invocar un servicio, comprende la transmisión de la solicitud por la red y la respuesta resultante, pero no incluye al software que presta el servicio real.

En el caso del WorkFlow, el cliente serían las aplicaciones Web, mientras que el servidor brindaría el servicio de workflow. El mecanismo de comunicación entre ellos (middleware) se hará utilizando colas de mensajes (MSMQ) como se indica en la Figura 5. Dicho mecanismo será explicado con más detalle en secciones posteriores. (Apéndice de Comunicación Cliente/Servidor).

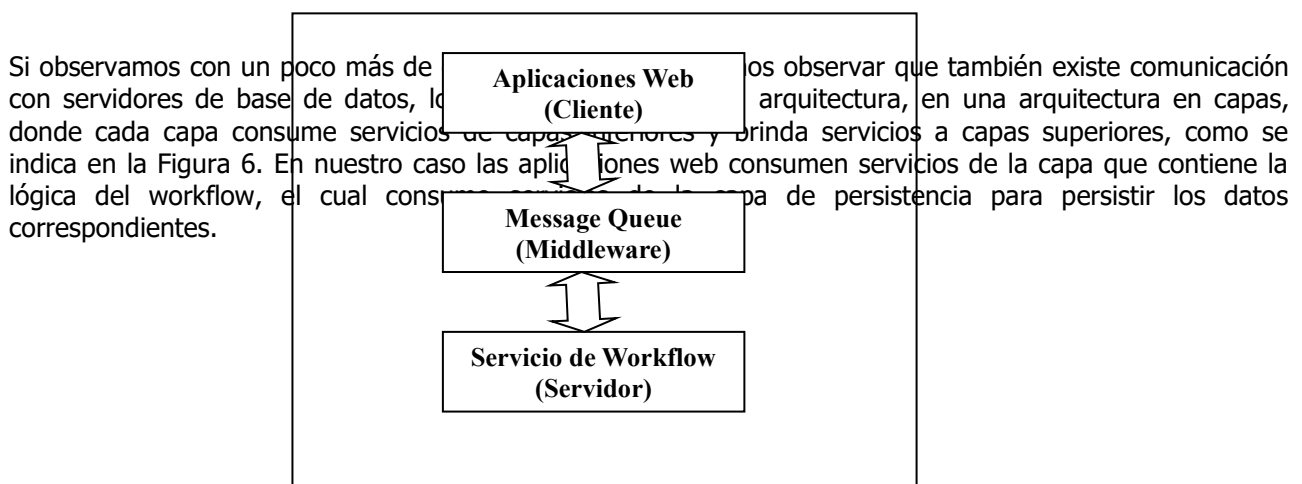


Figura 5

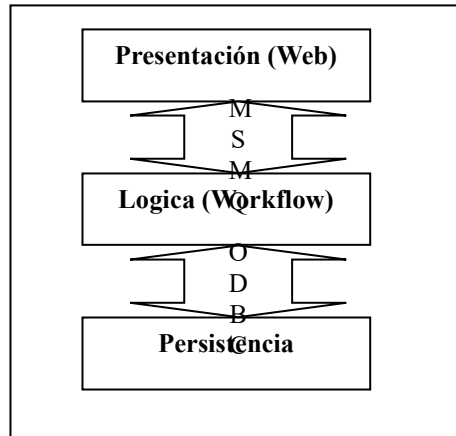


Figura 6

A continuación se enumeraran algunas de las ventajas de una arquitectura en capas:

- Una capa se independiza del resto si se utilizan interfaces estándar.
- Facilita el desarrollo en equipo.
- Es sencillo reutilizar componentes, sobre todo de las capas inferiores.
- Facilita el deploy en ambientes distribuidos.
- Separación de intereses entre capas.
- Escalabilidad, ya que el número de clientes es transparente para el servidor.

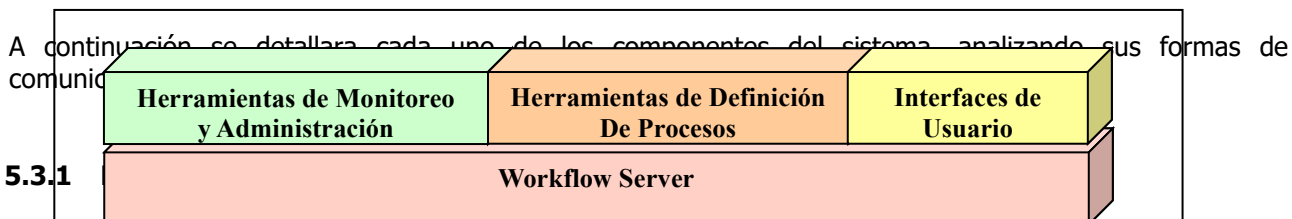
En contrapartida esta arquitectura presenta los siguientes inconvenientes:

- Tiende a degradar la performance
- Puede ser complicado encontrar los niveles de abstracción adecuados.
- Modificar la firma de un servicio afecta a los clientes

Por las ventajas anteriormente enumeradas, y por los requerimientos del sistema, el cual cuenta con numerosos clientes, carga variada en los servidores de aplicación y base de datos, etc. se ha considerado que la arquitectura en capas es la más adecuada.

5.3 Componentes principales.

En la sección anterior se señalaron los principales componentes de un sistema de workflow. En esta sección, se presentaran los principales componentes de nuestro sistema, los cuales tienen gran similitud con los ya vistos.



Este componente, es una herramienta gráfica utilizada para la creación de los procesos de negocio que posteriormente ejecutará el workflow. Dicha herramienta permite la construcción gráfica de los procesos, dibujando la red de actividades y su interrelación. Figura 7 fuera un grafo, en donde, existe la posibilidad de asignar participantes a determinados tipos. El resultado de esta herramienta es un archivo XML con la especificación del proceso creado. Esta herramienta está basada en un lenguaje formal de definición de procesos, es decir, el archivo XML que se genera sigue una cierta especificación.

El lenguaje utilizado para la representación de la definición de procesos se llama XPD (XML Process Definition Language), y es el lenguaje definido por la WFM (Workflow Management Coalition). Por más información sobre el lenguaje XPD, ver Apéndice - Interfase 1 (Marco Teórico).

5.3.2 Interfaz de Usuario.

Son las interfaces que se le presentan a los usuarios, y que indican, por ejemplo, su lista de tareas a realizar, formularios de ingreso de datos necesarios en un proceso, etc.

Muchos de los procesos, necesitan el ingreso de información por parte de un determinado usuario del workflow. La información que el mismo debe ingresar, es señalada en la definición del proceso, por este motivo se realizó un componente que genera una interfaz de forma automática, según la información especificada. También se realizó un componente que permita especificar la información a ingresar por el usuario, de forma que dicha especificación pueda ser interpretada por el componente anteriormente descrito. Ver apéndice - Administración de la información (Solución Detallada).

5.3.3 Herramientas de Administración y monitoreo

Con estas herramientas se pueden administrar los procesos corriendo en el server, se puede ver cuáles son los procesos corriendo, ver en qué estado están, se los puede pausar, parar, reiniciar y resumir un proceso que haya sido previamente pausado.

5.3.4 Workflow Server

El Workflow Server contiene, entre otros, el motor del Workflow. Es este el encargado de la ejecución de todos y cada una de las instancias de los procesos, también debe brindar las funcionalidades necesarias para monitorear y supervisar los procesos, para manejar y distribuir participantes, manejar sesiones, e interpretar las definiciones de proceso.

Cómo se puede apreciar, el servicio de workflow presenta varias funcionalidades, lo que hace de este componente, un sistema bastante complejo, detallado más adelante en este mismo documento.

5.4 Diseño de los componentes.

La siguiente imagen presenta el diseño de los componentes anteriormente detallados, viendo sus principales módulos.

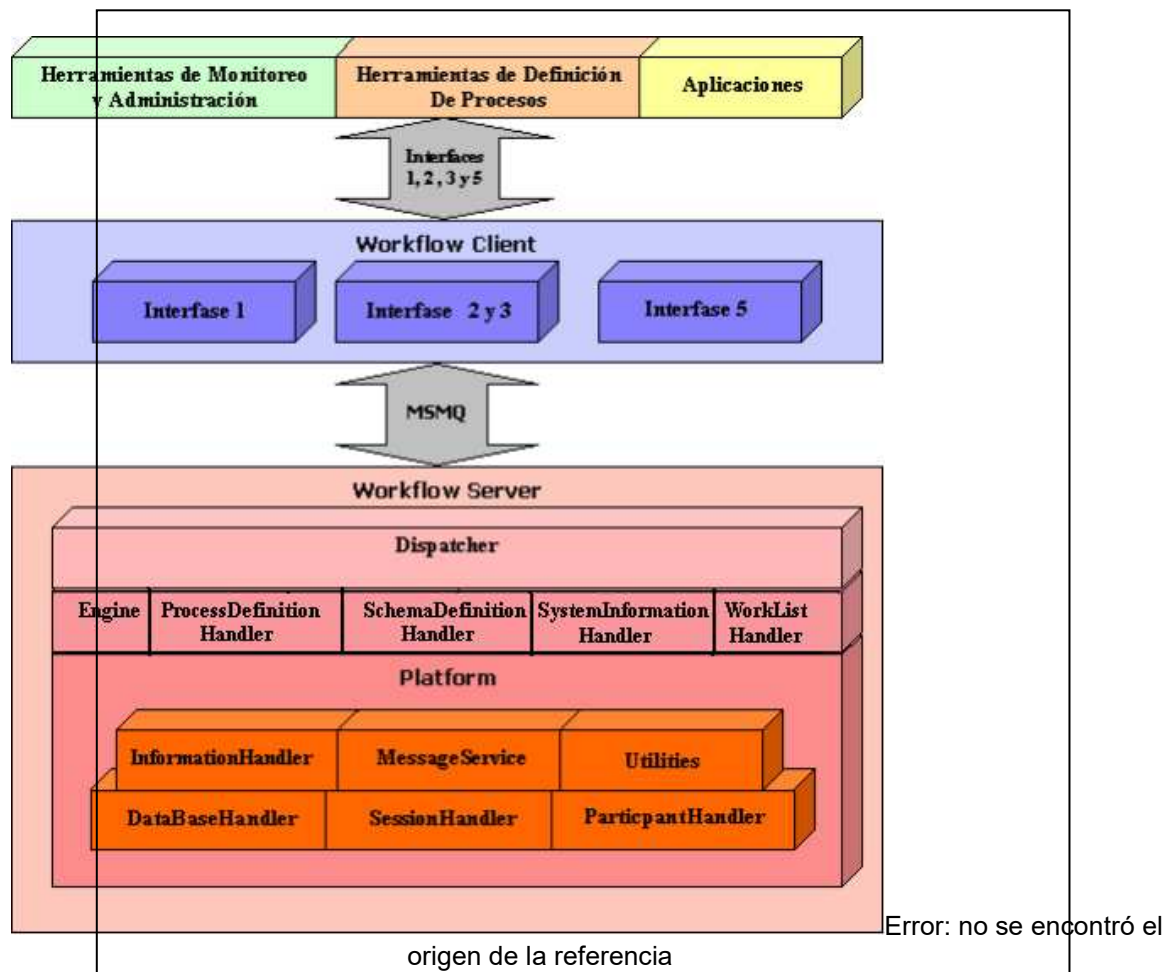


Figura 8

5.4.1 Workflow Client

Este componente es el encargado de comunicarse con el Workflow Server, y brinda una interfase que permite desarrollar todas las herramientas necesarias, ya sean administrativas, de consulta, de monitoreo y la herramienta de definición de procesos o esquemas. Todas encontrarán en esta interfaz, las funcionalidades necesarias. De esta forma, al desarrollar una de estas herramientas, solamente se debe tener conocimiento y utilizar el Workflow Client, sin preocuparse por como está desarrollado o que particularidades técnicas tiene el Server (ni su ubicación, ni sus interfaces, ni como comunicarnos con el).

Cuando se realice la distribución del sistema, se debe tener en cuenta que tanto las herramientas como el Workflow Client deben estar en la misma máquina, ya sea el servidor Web, o en cada estación de trabajo, si las aplicaciones se hacen del estilo WinForms, que no es el caso implementado en el prototipo, pero se podrían desarrollar.

La clase ICWorkFlowClient, es la cara visible de nuestro Motor de WorkFlow. Es el cliente, por lo que encapsula todas las llamadas al sistema. Todas las llamadas, sea cual sea el fin de las mismas, que necesiten interactuar con el motor de WorkFlow lo harán a través del cliente. Básicamente, el cliente presenta una serie de métodos que exponen todas las funcionalidades que el Motor de WorkFlow brinda. Las mismas se podrían agrupar por funcionalidades en los siguientes subgrupos:

- Conexión y desconexión.
- Administración de definiciones de procesos.
- Administración de definiciones de esquemas.
- Administración de instancias de procesos (Process Instances).
- Administración de instancias de actividades (Activity Instances).
- Administración de WorkItems.

Todas esas funcionalidades que exporta el cliente, en definitiva, publican todas aquellas funcionalidades que el WorkFlow brinda. Por tanto, cada una de estas funcionalidades, se mapea a alguna idéntica que brinda el server. El cliente, ante cada solicitud, realiza una llamada al Dispatcher. Como hemos visto en la descripción de la interacción entre el cliente y el Motor, las llamadas se realizan por la vía de intercambio de mensajes en colas de Message Queuing. En definitiva, cualquier operación brindada por el cliente, se traduce en generar el mensaje adecuado y colocarlo en la cola de entrada, de la que lee el Dispatcher. Para colocarlo en las colas, cuenta con un método privado llamado CallDispatcher (en alusión a que será el Dispatcher el que luego leerá de esas colas), que realiza dicha tarea.

CallDispatcher: es un método privado del cliente, que encapsula las llamadas al servidor. El Cliente le debe indicar al servidor cuando lo invoca la funcionalidad que está invocando, los datos (podríamos decir parámetros), y un indicador de si espera o no respuesta por esa llamada. Los parámetros los pasaremos en un Hashtable de .NET. En definitiva, crea un mensaje de MSMQ, pasando en el "sujeto" del mensaje el nombre de la funcionalidad, en el cuerpo la serialización de un Hashtable con los parámetros de la llamada.

5.4.2 Workflow Server

Como se puede ver en la Figura 8, el workflow server está compuesto por diferentes módulos: Dispatcher, Engine, Worklist Handler, ProcessDefinition Handler, SchemaDefinition Handler, SystemInformation Handler y Platform. Dichos módulos son los de mayor relevancia y serán detallados a continuación para comprender las funcionalidades que cada uno brinda y su interacción.

Dispatcher: El Dispatcher es el encargado de interactuar con el Cliente, utilizando el mecanismo que corresponda, en el caso del prototipo MSMQ. Este, se encarga de interpretar los mensajes enviados por el cliente desde la cola de entrada. La interpretación del mensaje se hace en función del subject. A partir de él, resuelve el componente del server que debe ser invocado y qué método dentro de cada uno de ellos. Los componentes que invoca son:

- Engine
- Worklist Handler
- ProcessDefinition Handler
- SchemaDefinition Handler
- SystemInformation Handler

A su vez, para aquellas invocaciones que requieran respuesta, envía el resultado a la cola de salida (ICWorkflow_Out). En los casos en que la ejecución en el servidor cancele por algún tipo de error, se generará una excepción que el Dispatcher captura. En estos casos, envía un mensaje a la cola de error, conteniendo información de la llamada (método y parámetros) y del error (mensaje de la excepción). Ver Apéndice - Comunicación Cliente – Servidor (Solución Detallada).

Engine: El motor provee funciones operacionales que permiten la ejecución (de instancias) de procesos de negocios, basadas en definición de procesos. Entre estas funciones destacamos:

- Interpretación de la definición de procesos
- Creación de instancias de procesos y gestión de su ejecución, incluyendo iniciar / detener / pausar / resumir.
- Navegación entre actividades y creación de los work items apropiados (utilizando el Worklist Handler)
- Funciones de supervisión y gestión

El funcionamiento del motor será detallado en secciones posteriores

ProcessDefinition Handler: Este componente es el encargado de gestionar las definiciones de procesos, brindando las funcionalidades necesarias:

- Obtener una definición según su identificador
- Modificar una definición de proceso
- Eliminar una definición de proceso
- Listar las definiciones de proceso
- Crear una definición de proceso

SchemaDefinition Handler: Es el componente encargado de gestionar las definiciones de esquemas, que utilizan los diferentes procesos:

- Agregar un esquema
- Eliminar un esquema
- Modificar un esquema
- Obtener un esquema
- Listar los esquemas existentes.

SystemInformation Handler: Este componente maneja información relevante al sistema, como ser la navegabilidad, las páginas web, etc.

- Lista los sistemas existentes
- Lista las páginas webs
- Lista los menús

Worklist Handler: Una de las principales funcionalidades que debe brindar un workflow es la de asignar recursos a diferentes actividades. Para comprender mejor el funcionamiento del Worklist Handler, definiremos algunas de las entidades anteriormente mencionadas:

Workitem: Es la representación del trabajo a ser procesado (por un participante) en el contexto de una actividad de una instancia de proceso.

Worklist: Es una lista de Workitems asociada con un determinado participante. La WorkList forma parte de la interfase entre el motor de Workflow y el WorkList Handler.

Participant: Un participant es un recurso que realiza una determinada tarea. Esta tarea es manifestada como uno o más Workitem asignados a un participante mediante la worklist. El término participante es generalmente utilizado para humanos, pero podrían ser máquinas, agentes inteligentes, etc.

El Worklist Handler, es uno de los componentes principales del Workflow, ya que el manejo de recursos y su asignación a las diferentes tareas, es una de las principales características de un sistema de Workflow. Es el encargado de proveer todas las funcionalidades sobre las Worklist, de todos los recursos, y es el único componente del sistema que va a interactuar con estas. El Worklist Handler es el encargado de asignar, eliminar, modificar, la lista de trabajo de un participante del workflow. Por más información ver apéndice - Manejador de lista de tareas (Solución Detallada).

A continuación enumeramos las funcionalidades que brinda el worklist Handler:

- Lista los workitems de un participante
- Crea un workitem
- Completa un workitem
- Detiene un worktitem
- Reasigna un workitem
- Obtiene un workitem

Platform: Involucra un conjunto de módulos y funcionalidades totalmente genéricas que son utilizadas por el sistema de Workflow pero están realizadas para que puedan ser utilizadas por cualquier sistema. Entre estos módulos podemos destacar:

DataBase Handler: Es una capa abstracta intermedia de acceso a la información. Brinda funcionalidades de acceso a datos, abstrayendo el tipo de sistema de almacenamiento utilizado. Por más información ver apéndice - Componentes Reusables (Solución Detallada).

Session Handler: Nos permite manejar sesiones de usuarios y validarlos. Es el encargado de gestionar las sesiones de los usuarios, con funcionalidades de altas y bajas. A su vez valida las sesiones de los usuarios, cuando estos realizan operaciones sobre el WorkFlow. Por más información, ver apéndice - Componentes Reusables (Solución Detallada).

MessageService: Es el servicio que nos permite levantar las colas de mensajes, para el correcto funcionamiento del sistema. Es un servicio genérico que permite levantar cualquier cantidad de

procesos que consuman mensajes de distintas colas de Message Queue. Ver - apéndice Comunicación Cliente-Servidor (Solución Detallada).

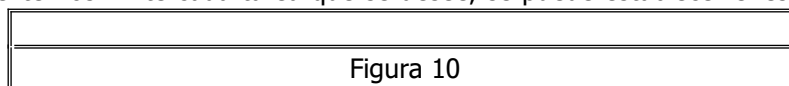
Participant Handler: Brinda funcionalidades de manejo de participantes, ya sea altas, bajas, modificaciones o consultas, permitiendo acceder a participantes que estén ingresados en el sistema construido, o a otros repositorios ya existentes en el cliente. Ver apéndice - Gestor genérico de participantes y apéndice sobre Componentes Reusables (Solución Detallada).

Utilities: Brinda un conjunto de funcionalidades para manejar XML, Reflection, Colecciones genéricas, HashTables extendidos, etc. Dichas funcionalidades fueron pensadas para poder ser reutilizadas en cualquier proyecto, de forma de comenzar a generar una plataforma de aplicaciones genéricas en la empresa cliente.

Information Handler: Presenta el concepto de esquema como unidad lógica, por medio de la cual administramos la información. A través de un Workflow, viaja información, que debe ser estructurada, presentada al usuario, modificada por éste o por procesos externos al Workflow, almacenada en fuentes de datos, etc., por lo que hemos decidido utilizar el concepto de "Esquema" para representar dicha información. Básicamente, un esquema (en nuestra solución) es una extensión del concepto de esquema relacional clásico, al que hemos agregado una serie de propiedades a tablas, columnas y relaciones, que nos permiten administrar de forma genérica la información contenida en ellos, tanto a la hora de presentarla al usuario, como de actualizar la fuente de datos con cambios que se hayan realizado sobre el mismo. Este proyecto presenta el esquema y herramientas para definirlo (por parte de usuarios administradores del Workflow), para generar formularios Web adecuados (de forma dinámica) para presentar la información al usuario y para impactar cambios en sobre la información contenida en una instancia, en la fuente de datos respectiva. El InformationHandler como tal, consta por tanto de tres partes fundamentales:

- **SchemaDefinition:** Esta herramienta brinda todas las funcionalidades necesarias para definir un "esquema", tanto desde el punto de vista de su estructura relacional (tablas, columnas, tipos, relaciones, restricciones, claves primarias, etc.), como una serie de propiedades que se definen sobre tablas y columnas, que serán utilizadas por las herramientas que veremos a continuación.
- **SchemaWeb:** Una política probadamente efectiva y bien vista, generalmente adoptada en el desarrollo de aplicaciones Web con .NET Framework, es la de desarrollar controles Web, en lugar de páginas. Luego, es posible generar toda una jerarquía de controles, unos dentro de otros, e incluso muchos de éstos en una única página Web. El objetivo del SchemaWeb, es generar un control Web dinámico que se "sepa", a partir de un esquema, autogenerarse y presentarse en pantalla. Un control (SchemaWeb_Main) es el que realiza la mayor parte de la tarea a la hora de desplegar el formulario en pantalla. Cada vez que debe desplegar el contenido de una tabla con varios registros, lo hace en forma de grilla, para lo cual se vale del control que la tabla tenga establecido, o en su defecto, un control genérico (SchemaWeb_Grid). Cuando debe desplegar el contenido de un único registro, lo hace mediante un formulario, utilizando el control establecido o un control genérico predefinido (SchemaWeb_Form).
- **SchemaImpact:** Esta herramienta, utilizando las potencialidades del conjunto de datos (DataSet) y adaptador de datos (DataAdapter) del Framework, impacta los cambios experimentados por el contenido de una instancia de un esquema, en una determinada fuente de datos. Consta básicamente de dos etapas; una primera en que resuelve el orden adecuado en el que impactar las tablas, de acuerdo a la estructura relacional del esquema (tomando en cuenta la integridad de las foreign keys a nivel de base de datos) y una segunda en la que impacta cada una de las tablas en el orden establecido, utilizando propiedades de las mismas, que permiten definir de forma automática las sentencias de insert, update y delete, para cada uno de los registros.

Con las tres herramientas juntas, se cubren todas las necesidades relacionadas con la interacción con usuario o procesos externos. Ante cada tarea que se desee, se puede establecer el esquema que seguirá la información que manejará la misma. El esquema se define utilizando la primera



herramienta (SchemaDefinition). Cuando el motor llega a esta tarea, sabiendo que tiene un esquema asociado, lo recupera y se lo pasa a la segunda herramienta (SchemaWeb), que a partir de éste, es capaz de desplegar una interfaz web adecuada para la visualización o ingreso de la información. La siguiente tarea, posterior a una que modifique la información, que quien define el proceso coloque podría perfectamente ser una instancia de la tercer herramienta (SchemaImpact), pasándole el esquema que sigue la información y el XML con los datos. SchemaImpact, es capaz de identificar los cambios que el esquema ha experimentado desde la última sincronización con la fuente de datos e impactarlos todos en la fuente.

En el apéndice M -- Administración de la Información, se puede estudiar con mayor detalle el concepto de esquema, su justificación teórica, la potencialidad de su definición y cada una de las tres herramientas anteriormente presentadas.

5.4.3 Comunicación entre servidor y cliente de Workflow.

Debido a la necesidad de separar el Server y el Client por escalabilidad y performance, es que se ha decidido que estos se comuniquen remotamente. Para lograrlo, existen varios mecanismos como ser remoting, corba, RPC, MSMQ, etc. Se analizaron las diferentes opciones y se optó por utilizar MSMQ, si bien, probablemente no sea tan performante como utilizar remoting, brinda la posibilidad de no perder mensajes cuando el Server esté fuera de servicio. La forma de comunicación entre estos podría ser realizada de una manera diferente a la propuesta, y queda pendiente la implementación de un prototipo utilizando otro mecanismo (un servidor exponiendo los métodos como WebServices y un cliente Java que desde una plataforma Linux acceda y consuma esos servicios).

Como se ha visto antes, el sistema está formado por una serie de componentes básicos. Entre éstos encontramos herramientas de monitoreo y administración, herramientas de definición de procesos, aplicaciones para el cliente, el cliente de WorkFlow y servidor de WorkFlow. El siguiente diagrama, presenta de forma gráfica, los anteriores conceptos.

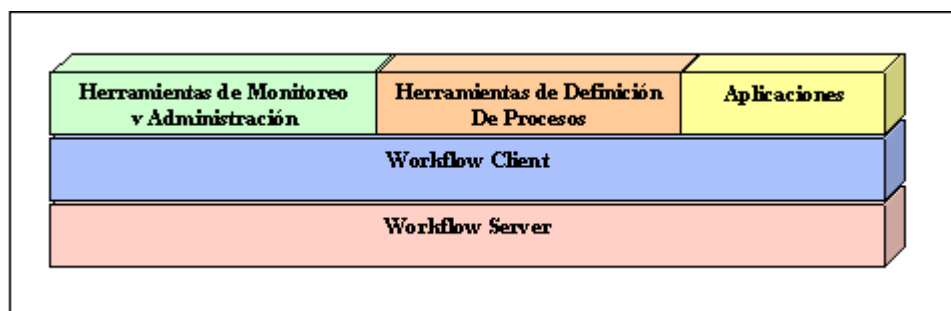


Figura 9

En la presente sección analizaremos el cliente de WorkFlow (WorkFlow Client) y su comunicación con el servidor, por lo que podemos ver con un poco más de detalle en el siguiente diagrama donde se centrará nuestro estudio. Vemos el diagrama anterior, con un poco más de detalle y contrastando en colores nuestro objetivo de estudio, contra una tonalidad gris del resto del sistema, que no nos convoca en esta sección del documento.

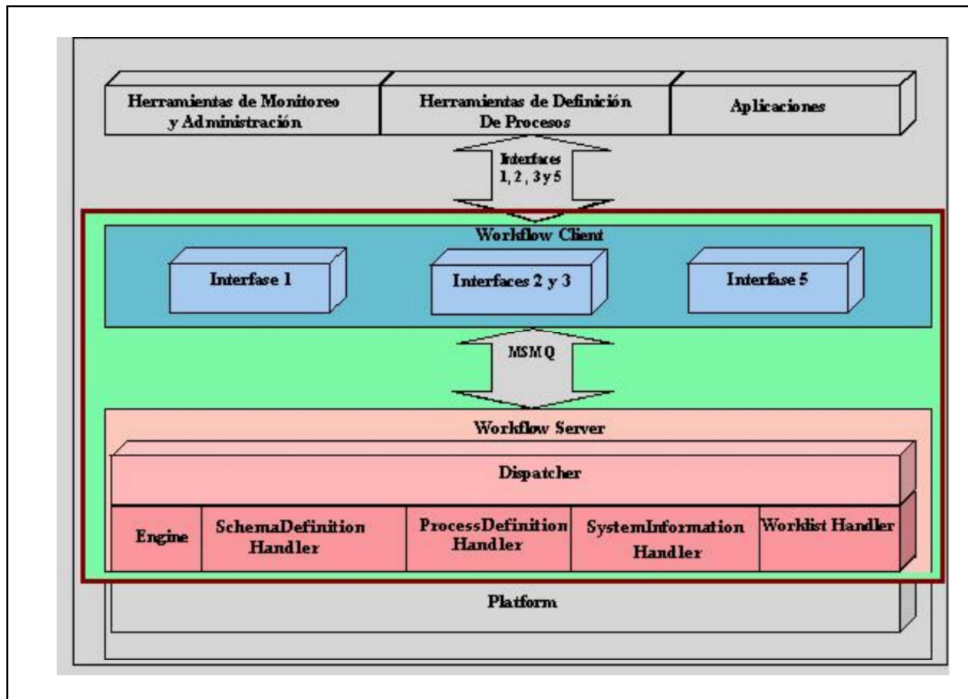


Figura 10

Como vemos, los componentes que estudiaremos son:

- Workflow Client (presenta hacia el exterior las interfaces de nuestro sistema).
- Workflow Server (Dispatcher, Server (Engine, ProcessDefinitionHandler, SchemasDefinitionHandler, SystemInformationHandler, WorkList Handler)).
- Comunicación entre éstos vía Message Queuing. Analizaremos principalmente el servicio de administración de mensajería que utilizamos a fin de administrar el comportamiento de las colas de MSMQ.

Dentro del Workflow Server, detallaremos principalmente la labor del Dispatcher, que es quien recibe y redirige las solicitudes a los demás componentes del servidor de Workflow.

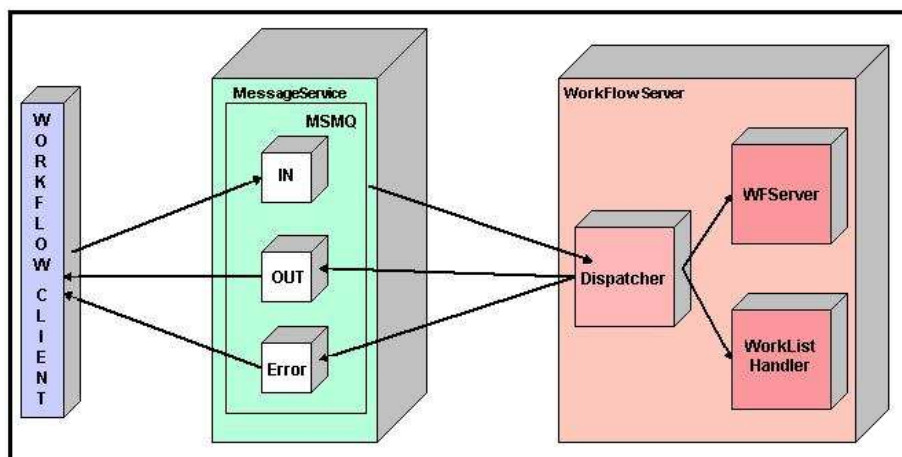


Figura 11

Al ir a un nivel mayor de detalle, se debe diferenciar dentro del servidor los componentes principales, que serían el motor de WorkFlow (WorkFlowEngine), el administrador de las definiciones de procesos (ProcessDefinitionHandler), el administrador de definición de esquemas (SchemaDefinitionsHandler) y el administrador de información del sistema (SystemInformationHandler).

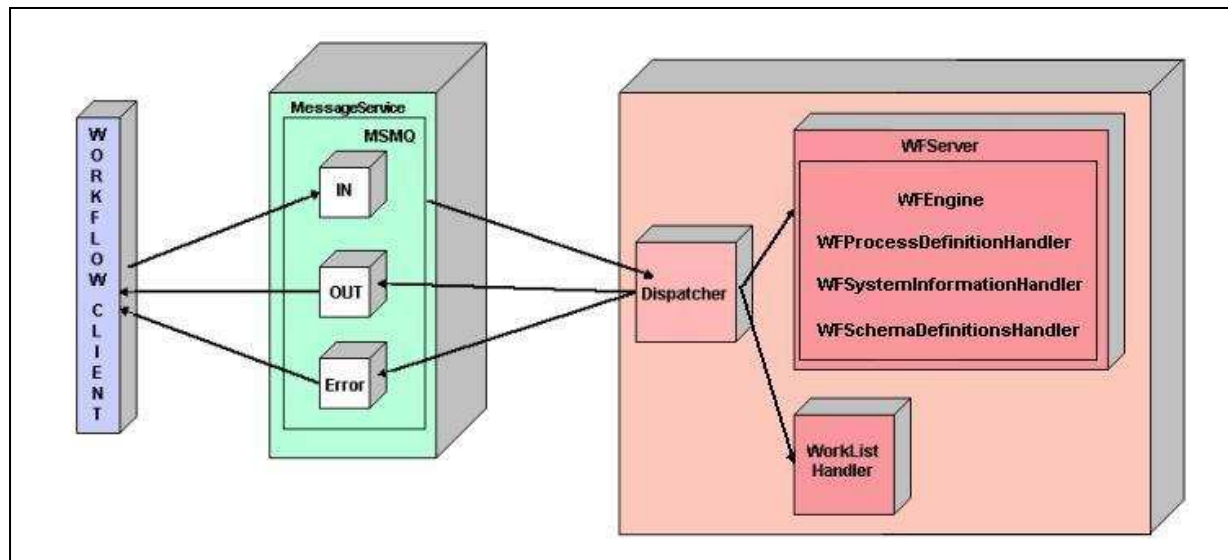


Figura 12

Como se observa en los anteriores diagramas, la comunicación entre cliente y servidor se lleva a cabo vía mensajería de Message Queuing. Message Queue ofrece un servicio de mensajería en el que se definen colas de mensajes, por medio de las cuales se pueden intercambiar mensajes. Una cola, es en definitiva un repositorio al cual varias aplicaciones pueden mandar mensajes y del cual alguna aplicación leerá los mismos. Un mensaje de MSMQ permite definir un identificador, un asunto y un cuerpo de mensaje. A través de éstos, las aplicaciones pueden comunicarse de forma relativamente transparente. Este es uno de los motivos por los cuales hemos seleccionado MSMQ como mecanismo de comunicación entre el cliente y el servidor, dado que si en un futuro, cualquiera de ellos debe ser modificado, estos cambios no necesariamente impactarán en el otro. Por ejemplo, podríamos fácilmente desarrollar un cliente basado en WebServices, es decir, podemos desarrollar un servicio Web con varios métodos que expongan todas las funcionalidades del WorkFlow, y comunicarnos detrás de estos servicios con MSMQ, con lo cual ni el Dispatcher, ni el resto del servidor notaría la diferencia en el cliente.

La idea ha sido definir tres colas de mensajes. Una cola llamada IN será donde el cliente volcará sus mensajes de solicitud al servidor. Éste recogerá los mensajes, actuará en consecuencia y responderá al cliente por una segunda cola, llamada OUT. La tercera cola es una cola de ERROR, donde el servidor volcará mensajes con la identificación de las llamadas que han dado un error en el servidor, la descripción del mismo y algún dato más que resulte de interés. En un principio, si bien el servidor volcará mensajes a la cola de error, no se realizará ningún tipo de operación con éstos. La cola se ha definido por extensibilidad, pensando en implementar en un futuro un cliente más inteligente que practique operativas de reintento o recuperación al estilo de lo que realiza BizTalk Server, que cuando recibe un error, espera un tiempo y vuelve a reintentar algunas veces antes de notificar el error al usuario. Para que esto sea posible, lo mínimo que necesitamos en el mensaje de error es la llamada realizada, el identificador de la misma y los parámetros para poder reintentar.

5.4.4 Funcionamiento del Motor.

Cómo ya se mencionó anteriormente, el motor de WorkFlow, es el encargado de gestionar la ejecución de procesos de negocio. Con tal motivo enumeraremos a continuación los métodos que brinda:

- **Start:** Este método crea una nueva instancia de un proceso de negocio, y comienza su ejecución.
- **Stop:** Finaliza un proceso de negocio, ni bien termine de ejecutarse la actividad actual.
- **Restart:** Reinicia la ejecución de un proceso de negocio.
- **Suspend:** Pausa la ejecución de un proceso de negocio.
- **Resume:** Continúa la ejecución de un proceso que había sido suspendido.
- **ListProcess:** Lista los procesos iniciados.

Para comprender el funcionamiento del motor, analizaremos la ejecución de un proceso, analizando los estados por los que pasa, la información que se mantiene, y el mecanismo para evaluar condiciones para la toma de decisiones.

Se comienza por mencionar los diferentes estados de un proceso y de sus respectivas actividades:

Estados de Proceso:

- **Open.Running:** Indica que el proceso está ejecutándose sin inconvenientes.
- **Open.NotRunning.NotStarted:** Indica que el proceso no comenzó a ejecutarse aún, ya que no fue iniciado, pero está en proceso.
- **Open.NotRunning.Suspended:** El proceso fue suspendido por un usuario.
- **Open.NotRunning.Suspended.byActivity:** El proceso se ha suspendido, por que la última actividad a ejecutar fue suspendida.
- **Open.NotRunning.Suspended.bySystem:** El proceso se ha suspendido externamente, por un usuario del sistema.
- **Closed.Terminated:** El proceso fue terminado manualmente por un usuario del sistema.
- **Closed.Aborted:** El proceso fue abortado por errores en la ejecución de actividades o toma de decisiones.
- **Closed.Completed:** El proceso terminó de ejecutarse correctamente.
- **Closed.Restarted:** El proceso fue terminado para ser reiniciado.

Estados de una actividad

- **Open.Running:** La actividad está siendo ejecutada en este momento.
- **Open.Suspended.byStartMode:** Significa que la actividad fue suspendida por que su comienzo debe ser realizado manualmente.
- **Open.Suspended.byExecution:** Significa que la actividad a ejecutar es asincrónica.
- **Open.Suspended.byFinishMode:** La actividad es suspendida por tener un final manual
- **Close.Aborted:** La actividad fue abortada por un error en su ejecución
- **Close.Terminated:** Indica que la actividad fue terminada por un usuario
- **Close.Completed:** La actividad fue terminada con éxito.

Ahora que se conocen los diferentes estados por los que pasa un proceso y sus actividades, analizaremos la información que maneja un proceso y sus actividades.

ProcessData: Mantiene toda la información del proceso, incluye:

- GlobalData:
 - Variables globales del proceso (Datafields)
- StateData:
 - Definición del esquema con el cual fue creado el proceso
 - Identificador de la instancia de proceso
 - Estado del proceso
 - Fecha de apertura
 - Resultado de ejecución de la última actividad
 - Estado de la ejecución de la última actividad
 - XML de entrada al proceso
 - Identificador de la actividad actual
- UserData:
 - Xml que ingresa el usuario en una actividad de tipo participant

ActivityData: Es la información referente a la actividad a ejecutar, incluye:

- GlobalData: Parámetros de la actividad.
- StateData:
 - Identificador de la instancia de la actividad.
 - Estado de la actividad.
 - Participantes involucrados en la actividad.
 - Hito en que se encuentra
 - Tiempo limite
 - Prioridad de ejecución de la actividad
 - Título de la actividad
 - Descripción de la actividad

ActivityContext: Es la información que reciben los toolAgents (aplicaciones externas):

- **ProcessData:** Debe recibir la información referente al proceso, ya que potencialmente puede, entre otras cosas, modificar el estado del mismo.
- **ActivityData:** Recibe toda la información referente a la actividad a ejecutar.
- **ActivityExecutionResult:** Contiene el resultado de la ejecución de la actividad.
- **SuspendExecution:** Indica si la ejecución debe ser suspendida porque la actividad es asíncrona.

Una vez comprendida la información que debe manejarse en una instancia de proceso y los estados por los que el mismo va pasando, se está en condiciones de presentar la ejecución de un proceso, la cual se detallará a continuación.

Ejecución de un proceso

Para iniciar un nuevo proceso de negocio el motor se crea una instancia del mismo según su definición, seteando su estado como "Open.NotRunning.NotStarted" y persistiendo dicha instancia en la base de datos. Busca la primera actividad del proceso, seteándola como actividad actual del mismo, y se ejecuta el proceso modificando su estado a "Open.Running".

Al ejecutar el proceso, el motor toma la actividad actual y verifica que no haya sido pausada o finalizada, en este caso, se ejecuta la actividad actual. Antes de su ejecución, debe traducir todos los parámetros que ésta reciba, y que estén dados en el lenguaje que permite realizar expresiones en función del globalData o de datos de ingresados al proceso, sustituyéndolos por su valor en la instancia del proceso (Por más información sobre dicho lenguaje y el mecanismo de traducción ver Apéndice - Lenguaje para referenciar información del proceso – Solución Detallada). Luego de ejecutar dicha actividad, verifica el estado en que terminó la misma. En caso de que el estado haya sido: "Closed.Aborted" o "Closed.Terminated", el proceso es finalizado anormalmente. El resultado de dicha actividad puede ser también suspendida ("Open.NotRunning.Suspended.byActivity"), en este caso suspende la ejecución del proceso, ya que la actividad es asíncrona y se debe esperar a que ésta termine.

Una vez ejecutada la actividad, se persiste nuevamente el proceso, de forma de almacenar los cambios realizados por su ejecución y se busca la nueva actividad a ejecutar. Para esto, se recorre la colección de transiciones salientes y se selecciona la primera que su condición evalúe en verdadero. Se toma la actividad final de esa transición como la siguiente a ejecutar repitiendo el procedimiento anteriormente mencionado. En caso que no haya transiciones salientes o ninguna evalúe en verdadero, el proceso aborta. La condición a evaluar, puede ser una expresión en función de variables globales o datos ingresados al proceso, según el lenguaje previamente mencionado, por tal motivo, debe traducirse la expresión antes de ser evaluada con los valores de la instancia actual del proceso. Por más información sobre el mecanismo para generar y evaluar expresiones ver apéndice - Componentes reusables.

Se describió anteriormente el proceso de ejecución de un flujo, se detallará ahora como se efectúa la ejecución de una actividad. Al ejecutar una nueva actividad, se crea una instancia de la misma, pasándole toda la información del proceso, seteando su estado como "Open.Running" y persistiéndola.

Si la actividad a ejecutar tiene un modo de comienzo manual, se suspende su ejecución, de lo contrario se ejecuta dependiendo del tipo de actividad:

- **Implementation**
 - **None:** La única actividad de este tipo es el Milestone o hito, ya que es la única que interpreta el motor por sí mismo, y lo que debe hacer es modificar el hito de la instancia del proceso.

- Tool: Este tipo de actividad invoca una aplicación externa (toolAgent). Para que dicha aplicación pueda acceder a la información de proceso debe cargarse el ActivityContext con la información anteriormente detallada, luego se invoca la aplicación (ver Apéndice - Tool Agents - Solución Detallada). Si la aplicación se ejecuta de forma asincrónica, el proceso debe suspenderse (por ejemplo, en el caso de una tarea que debe ser realizada por un usuario), en este caso el estado de la instancia de la actividad es "Open.Suspended.byExecution". Una vez terminada la ejecución, debe actualizarse el ProcessData, según los resultados brindados por la aplicación.
- Route: Esta actividad es "tonta", su finalidad es poder agregar condiciones en los flujos, de forma de representar gráficamente el camino a seguir.

Una vez ejecutada la actividad, si su estado no fue suspendido por ejecución o por comienzo manual y la misma no debe ser finalizada manualmente (Open.Suspended.byFinishMode), se culmina con la ejecución de la misma, seteando su estado a "Closed.Completed" y persistiéndola. Por más información sobre el motor, ver Apéndice - Motor del sistema de WorkFlow (Solución Detallada)..

En el siguiente diagrama se puede apreciar más claramente el mecanismo de ejecución de un proceso.

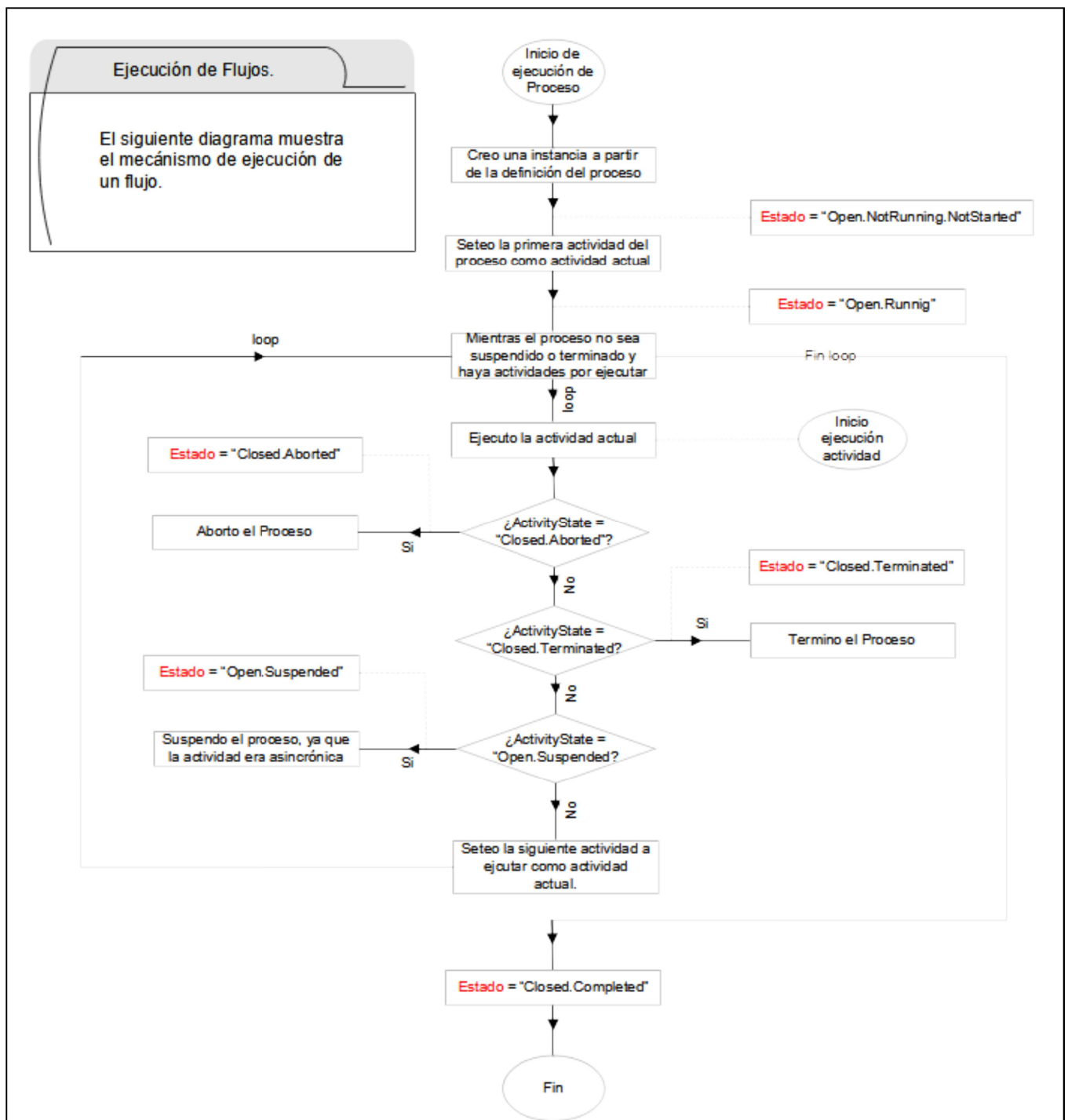


Figura 13

Se puede apreciar en el diagrama puede terminar o bien a causa de una actividad que se suspendió, aborto o fue parada, o bien, por que el propio proceso fue suspendido o terminado (de forma manual por un usuario). El diagrama que se presenta a continuación forma parte de la ejecución de un proceso y representa la ejecución de una actividad en particular, concluyendo de esta forma el flujo de ejecución de un proceso.

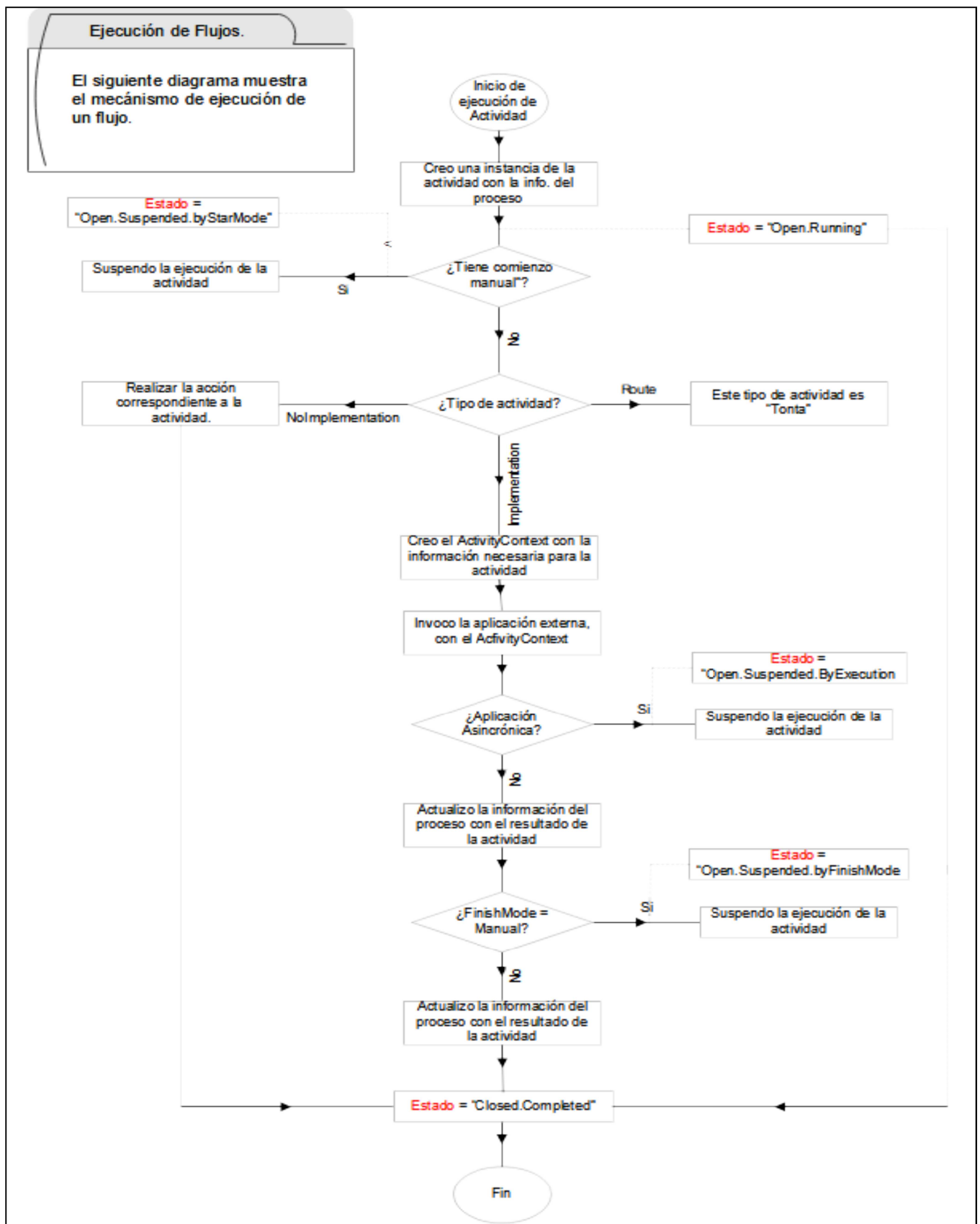


Figura 14

6 CASO DE USO – SOLICITUD DE LICENCIA.

En esta sección se enseñará un caso de uso, de forma tal que clarifique las ideas de los sistemas de workflow, viendo como se aplica a casos prácticos, y aprovechando para detallar como se implementa dicho proceso de negocio en nuestro sistema.

El proceso es simple. En una primera etapa un empleado ingresa su nombre, correo y período de solicitud de licencia. Cuando el empleado termine con esta etapa, se pasa la información a la lista de tareas pendientes del jefe directo. Cuando el jefe la consulte, puede seleccionarla para completarla. Completar la tarea pendiente para el jefe, es decidir si en función de las necesidades de su departamento, de las licencias ya adjudicadas a los compañeros, etc., puede o no autorizar la licencia del solicitante en ese período. El jefe indicará si la aprueba o no, pudiendo agregar un texto explicativo de los motivos que llevan a la decisión. Si el jefe directo, rechaza la licencia, el proceso termina y se comunica al correo electrónico del usuario la decisión y los motivos. En caso que el jefe lo apruebe, pasa al gerente. Nuevamente, al gerente le aparecerá una tarea pendiente en su lista. Cuando la seleccione para completarla, se le debe brindar la posibilidad de ver los datos del usuario, el período solicitado y la decisión del jefe de departamento. En función de estos datos y su visión más global que puede considerar por ejemplo, superposición de licencias con usuarios de otras áreas que no pueden permitirse, tomará la decisión final. Al igual que antes, la respuesta le llega al usuario solicitante por correo, a la dirección que el ha establecido.

6.1 Definición de esquemas.

Los esquemas de información administrada en los flujos los define un usuario administrador del WorkFlow. El objetivo es establecer el esquema que seguirá la información que fluye entre las distintas tareas del proceso, a la vez de establecer las visibilidades que, en cada etapa tendrá un usuario de dicha información. En este ejemplo particular, los datos que fluyen son los siguientes:

- Nombre del Solicitante
- Correo electrónico
- Fecha de Inicio
- Fecha de Finalización
- Indicador de aprobación del jefe directo
- Descripción ingresada por el jefe directo
- Indicador de aprobación del gerente
- Descripción ingresada por el gerente

El esquema de la información es siempre el mismo, lo que cambia de una tarea a la otra es la visibilidad y la posibilidad de modificar o no información de cada uno de los usuarios. La herramienta de definición de esquemas permite definir las propiedades de visibilidad y actualización sobre cada tabla y columna del esquema, lo que permite obtener este efecto sin mayores dificultades. Por mayor detalle sobre como definir esquemas y propiedades, y sobre como éstas influyen en la visualización de formularios y en el impacto de éstos en base de datos, dirigirse al Apéndice M: Administración de la información.

En la primera instancia, el usuario solicitante podrá ver:

- Nombre (modificable)
- Correo electrónico (modificable)
- Fecha Inicio (modificable)
- Fecha Finalización (modificable)

Paralelamente, no podrá ver ni modificar:

- Indicador de aprobación del jefe directo
- Descripción ingresada por el jefe directo
- Indicador de aprobación del gerente
- Descripción ingresada por el gerente

Siguiendo la misma lógica para la segunda etapa, cuando el jefe de departamento vaya a tomar su decisión debe poder observar, pero no modificar la información ingresada por el empleado:

- Nombre (visible – no modificable)
- Correo Electrónico (visible – no modificable)
- Fecha de Inicio (visible – no modificable)
- Fecha de Finalización (visible – no modificable)

A la vez, debe poder ingresar su decisión:

- Indicador de aprobación (visible – modificable)
- Descripción ingresada (visible – modificable)

No debe visualizar, y por consiguiente no modificar, los campos de la decisión del gerente.

- Indicador de aprobación del gerente
- Descripción ingresada por el gerente

De la misma forma el gerente debe poder visualizar, pero no modificar, el valor de los primeros seis campos, pero debe ingresar los valores que indiquen su solución en los últimos dos campos:

- Indicador de aprobación del gerente (visible – modificable)
- Descripción ingresada por el gerente (visible – modificable)

En principio, no parece demasiado significativo, el definir el esquema de la información en cada tarea de usuario, pero no hay que perder de vista que en función de este esquema el motor de workflow sabrá llamar al editor de formularios que, a partir de la definición del esquema y sus propiedades, presenta los formularios de forma adecuada.

Como ejemplo, se observa en la Figura 15, el formulario desplegado para el usuario solicitante.

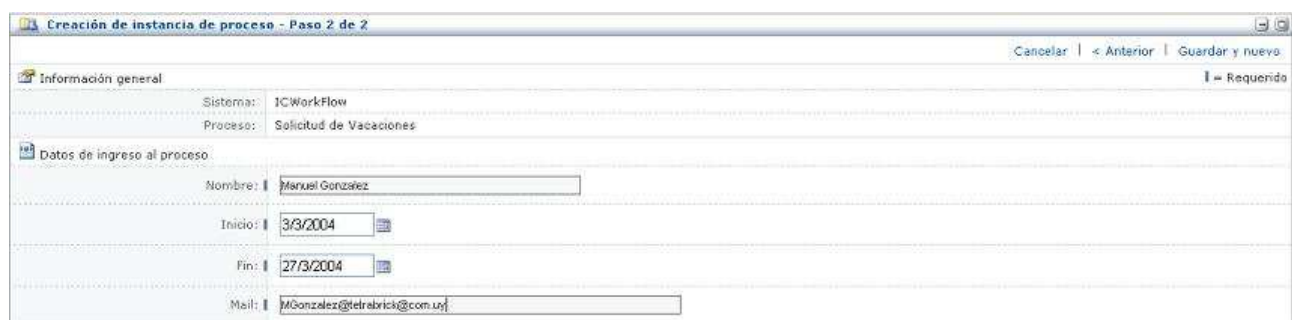


Figura 15

El formulario final, el visto por el último usuario (el gerente), es el indicado en la Figura 16.

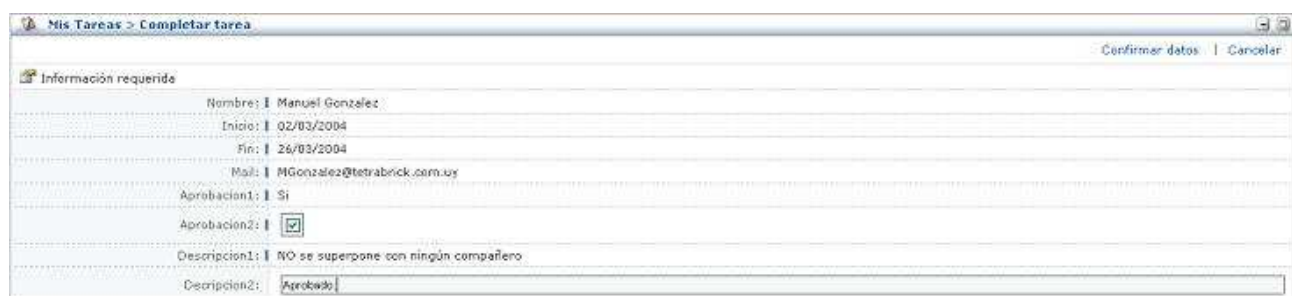


Figura 16

6.2 Definición de Procesos.

Una vez definido el esquema de la información que va a manejar el proceso de negocio, se está en condiciones de definir el mismo. Para esto, debemos utilizar la herramienta que permite definir un nuevo proceso, seleccionando la opción "Definición de procesos" dentro del menú "Administración", que se encuentra sobre la parte izquierda, como se muestra en la Figura 17.



Figura 17

Una vez seleccionada esta opción se pasa a una página en donde se muestran todas las definiciones de procesos existentes como muestra en la Figura 18, brindando posibilidades de edición, visualización, borrado y otra información relevante.

Definición de procesos								Nuevo		
Título	Esquema	Habilitado	Transaccional	Audita	Creado	Modificado				
Recuperación de deuda	Prueba	☒	☒	☒	02/09/2003	16/09/2003				
Aumento saldo en CC	Prueba	☒	☒	☒	12/09/2003	16/09/2003				
Test Gastón	Reclamos - Procesar reclamo	☒	☒	☒	22/09/2003	23/09/2003				
Recepción de reclamos	Reclamos - Datos del reclamo	☒	☐	☒	22/09/2003	30/09/2003				
Reclamo Call Center	Reclamo Call Center	☒	☐	☒	19/09/2003	22/09/2003				
Solicitud 2	Solicitud de Licencia	☐	☐	☐	25/05/2003	26/05/2003				
Atención al cliente	Servicio soporte Atención al cliente	☒	☒	☒	11/09/2003	20/09/2003				
Soporte Home Banking	Facturas de prueba	☒	☒	☒	11/09/2003	20/09/2003				
Oportunidades automotrices	Negocio automotriz	☒	☒	☒	11/09/2003	16/09/2003				
Prueba Rodrigo	Solicitud de Licencia	☒	☒	☒	24/05/2003	25/05/2003				
								← anterior	siguiente →	

Figura 18

En nuestro caso crearemos una nueva definición de proceso, para esto lo que se debe presionar, el botón "Nuevo" que se encuentra en la parte superior derecha de la página, accediendo de esta forma al editor de procesos presentado en la Figura 19.

Definición de proceso

Guardar y volver | Guardar y nuevo | Cancelar

Información general = Requerido

Sistema:

Título: Habilitado: ☐

Descripción:

Esquema: Ver

Opciones: ☐ Audita ☐ Transaccional

Diseño del proceso

GoDiagram, for evaluation only
Northwoods Software
www.nwoods.com

Inicio
Actividad
Hito
Condición
Fin

Prevista:

Transformación final

XSLT:

Validar XSLT

IC-Workflow Manager © Infocorp 2003 Version 2.0

Figura 19

Como se puede apreciar en la figura anterior, la página de creación de un proceso se divide en 3 partes, una parte en donde se define la información general del proceso, una parte en donde se diseña el proceso, y otra parte en donde se puede ingresar una transformación XSLT. A continuación se van a detallar cada una de las mismas.

6.2.1 Información general de un proceso

En la parte superior de la página de creación de un proceso es donde se deben a ingresar los datos generales del proceso que se está definiendo. Para esto se cuenta con una serie de campos en donde el usuario que crea el proceso debe completar con la información correspondiente. Estos campos son los que se detallan a continuación:

- Combo de selección del sistema al cual pertenece el proceso que se va a crear.
- Hay un campo llamado "Título" en donde se debe ingresar el nombre con el que se va a identificar el proceso.
- Checkbox que permite habilitar el proceso, para que este pueda ser ejecutado.
- Un área de texto en donde, si se desea, se puede ingresar una breve descripción sobre el proceso que se crea.
- Se cuenta con un combo, el cual se utiliza para seleccionar cual es el esquema de entrada al proceso. A partir de este esquema se va a generar el formulario de entrada al proceso.
- Checkbox que permiten señalar al proceso como auditable o transaccional (no disponibles)

Para el caso de la solicitud de vacaciones la información general del proceso se ingresa de la siguiente forma:

- El sistema al cual pertenece el proceso se llama "ICWorkflow".
- El título del proceso es "Solicitud de Vacaciones".
- Se marca el "checkbox" de "Habilitado".
- El esquema de entrada asociado al proceso es el llamado "Solicitud de licencia", que se definió previamente con el editor de esquemas.
- Se marca el "checkbox" para que se audite, pero no se marca el checkbox de "transaccionable".

Se puede ver en la siguiente Figura 20 como queda la parte de información general del proceso de solicitud de vacaciones con los datos anteriores ingresados.

La imagen muestra una interfaz de usuario con el título 'Información general' en la parte superior izquierda. En la esquina superior derecha hay un ícono de información y el texto '= Requerido'. El formulario contiene los siguientes campos:

- Sistema:** Un menú desplegable con 'ICWorkflow' seleccionado.
- Título:** Un campo de texto con 'Solicitud de Vacaciones' ingresado.
- Habilitado:** Un checkbox que está marcado.
- Descripción:** Un área de texto grande y vacía.
- Esquema:** Un menú desplegable con 'Solicitud de licencia' seleccionado, seguido de un botón 'Ver'.
- Opciones:** Dos checkboxes, 'Audita' (marcado) y 'Transaccional' (no marcado).

Figura 20

6.2.2 Diseño del proceso

En la Figura 21 se indica el control web que permite diseñar los flujos, que conjuntamente con la información general, conforman la definición del proceso de negocio.

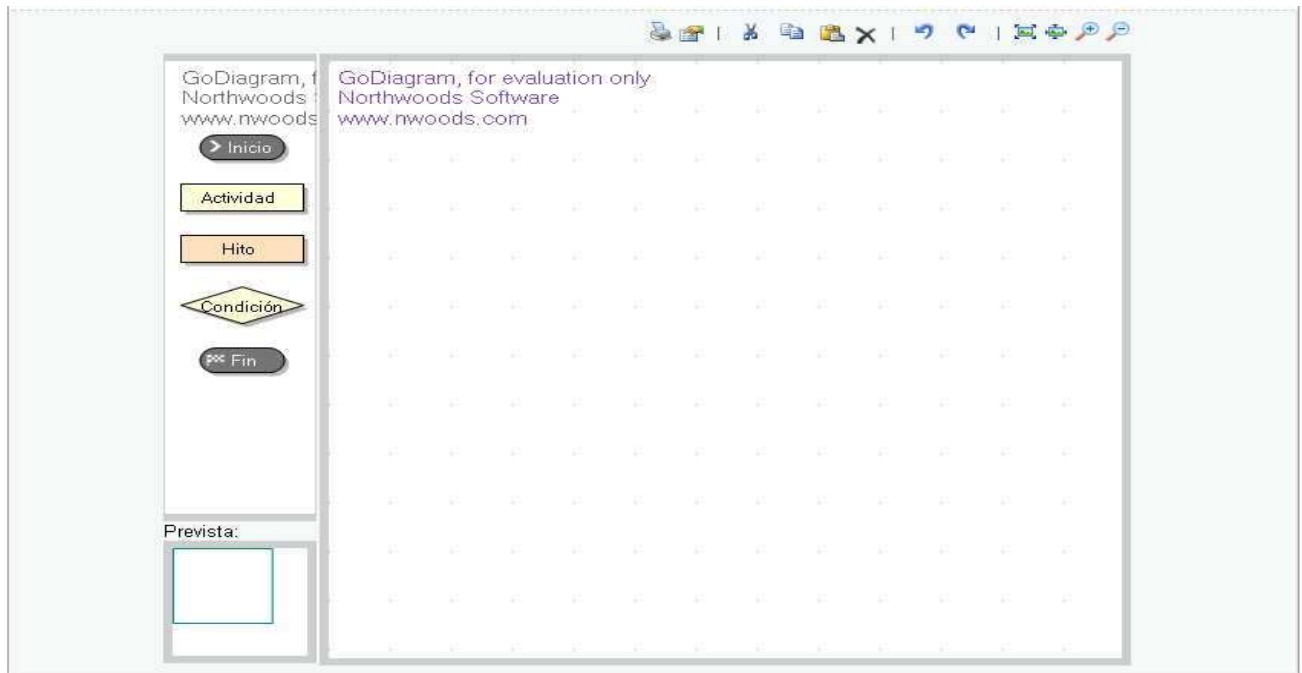


Figura 21

Es con este editor de flujos con el que se crean las actividades, decisiones y transiciones entre las mismas. Como se puede apreciar en la figura anterior, el editor de flujos se divide en 2 sectores. El sector de la izquierda, indicado en la Figura 22, presenta los distintos tipos de actividades que se pueden crear:



Figura 22

Mientras que el sector derecho del editor de flujos, se encuentra el área en donde se insertan las actividades y se dibuja el flujo, a esta área la llamaremos de ahora en adelante "área de diseño del flujo".

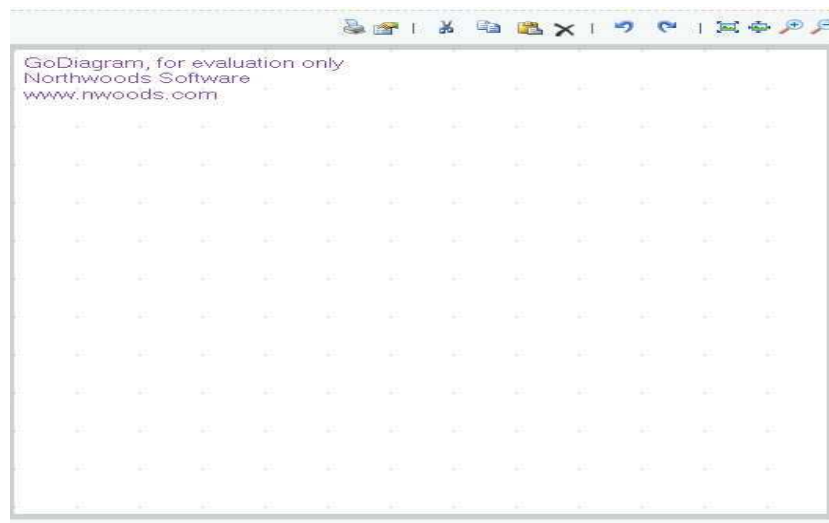


Figura 23

El mecanismo que se debe utilizar para crear un nuevo nodo es, primero seleccionar el tipo de nodo a crear, para esto se presiona el botón izquierdo del mouse sobre la parte izquierda del editor (Figura 21) y se elige el tipo de actividad, luego se vuelve a presionar el botón izquierdo del mouse sobre el "área de diseño del flujo" (Figura 22), en ese momento se abre automáticamente un formulario según el tipo de nodo seleccionado, en el cual se ingresan los datos correspondientes. Una vez completado dicho formulario se inserta la actividad en el "área de diseño del flujo".

Para definir las transiciones que van de una actividad a otra, se debe llevar el puntero del mouse sobre un lado del nodo origen de la transición arrastrando hasta el nodo destino, de esta forma se crea la transición que une dos actividades como se muestra en la Figura 24.

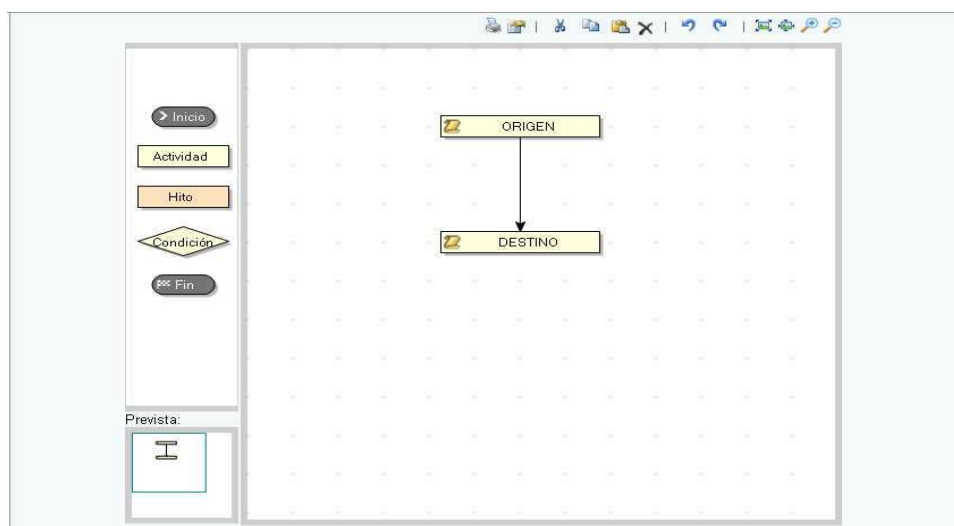


Figura 24

A continuación se detallan los distintos tipos de actividades que se pueden crear:

- Actividad "Inicio": Todo proceso debe tener una actividad de este tipo, en el momento de crear esta actividad, se abre el formulario indicado en la Figura 25.

Id	Nombre	Tipo	Valor

Figura 25.

En la parte llamada "Parámetros" es donde se definen las variables globales a un proceso.

- Actividad de tipo "Actividad": Este tipo de actividad comprende una amplia gama de actividades. Son todas aquellas que invocan aplicaciones externas al workflow. Dentro de esta gama de actividades se encuentran entre otras, la actividad que envía mail, la actividad que invoca "web services", etc. En el momento de agregar una actividad de este tipo sobre el "área de diseño del flujo", se abre un formulario en donde se puede seleccionar de una lista cual es la actividad que se desea crear.

Paso 1 de 2
El tipo asociado a la actividad define las características que esta tendrá, por lo cual en este paso se debe seleccionar el tipo de actividad a incluir.

Tipo:

- Milestone Change
- Participant Task
- Script Call
- Database Call
- Database Impact
- Webservice Call
- Assembly Call
- Send Mail
- Process Call
- Send Print

Figura 26

Una vez que se selecciona una actividad de la lista anterior, se presiona el botón siguiente del formulario, y se abre otro formulario en donde se presenta una breve descripción sobre la actividad seleccionada.

En la Figura 27 se muestra, a modo de ejemplo, como es la descripción cuando se ha seleccionado la actividad que manda mail, llamada "Send Mail".

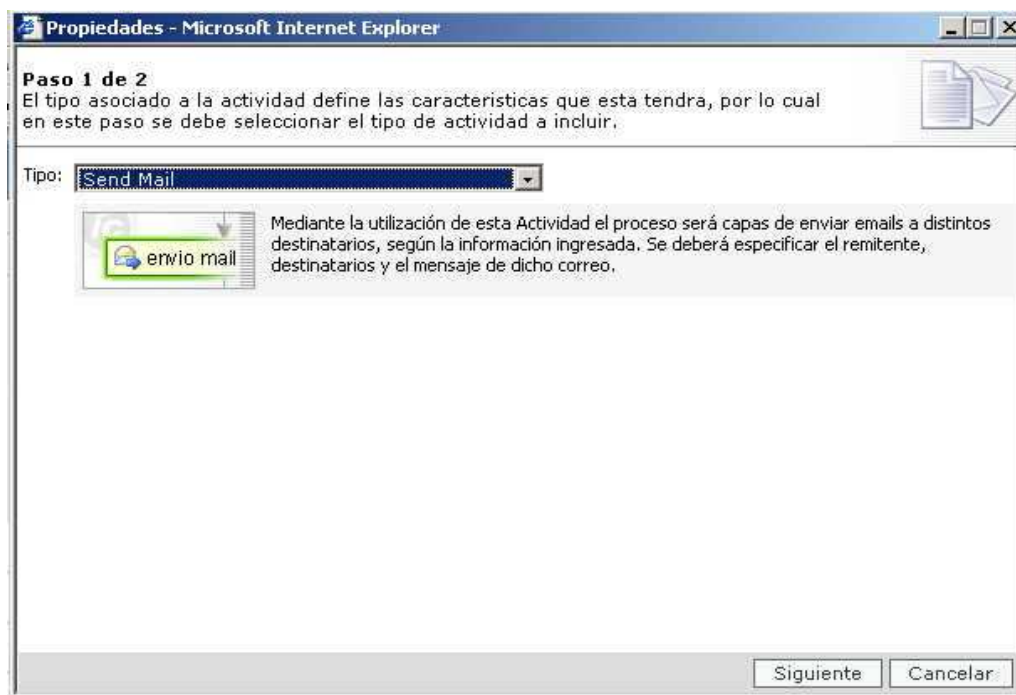


Figura 27

Finalmente se vuelve a presionar el botón siguiente y se abre un nuevo formulario en donde se ingresa la información de la actividad, los datos que se piden en este formulario varían según la actividad que se esté creando. A modo de ejemplo se muestra como es el formulario para el caso de la actividad "Send mail":

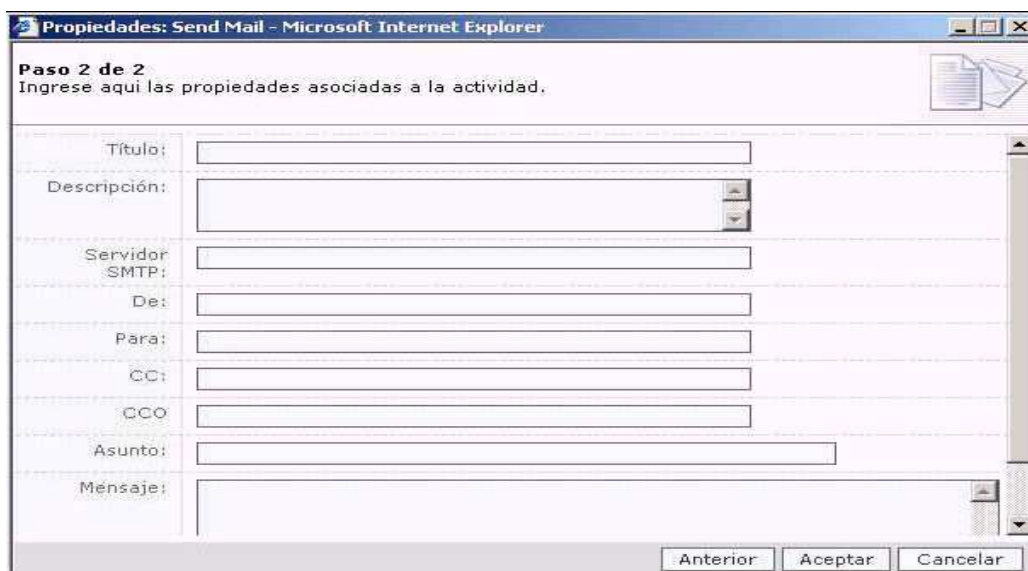


Figura 28

- Actividad "Condición": Este tipo de actividad es la que permite evaluar las condiciones que se ingresen y de acuerdo a dicha evaluación desviar el flujo hacia una actividad o hacia otra. En la siguiente figura se muestra como es el formulario que se debe completar para este tipo de actividad.

Id	Campo	Operador	Valor	Y/O
Agregar				

Figura 29

Como se puede apreciar en la Figura 29, el formulario cuenta con los siguientes campos:

- Un campo llamado "Título" en donde se ingresa el texto con el que va a aparecer esta actividad en el "área de creación del flujo".
- Un campo en donde si se desea se puede escribir una breve descripción sobre la actividad de condición.
- Un área en donde se ingresa una condición a evaluar.

- Actividad "Hito": Este tipo de actividad sirve a modo informativo y se utiliza por lo general para marcar cual es el estado del proceso ese momento. En la Figura 30 se muestra como es el formulario que se debe completar en el momento de crear una actividad de este tipo.

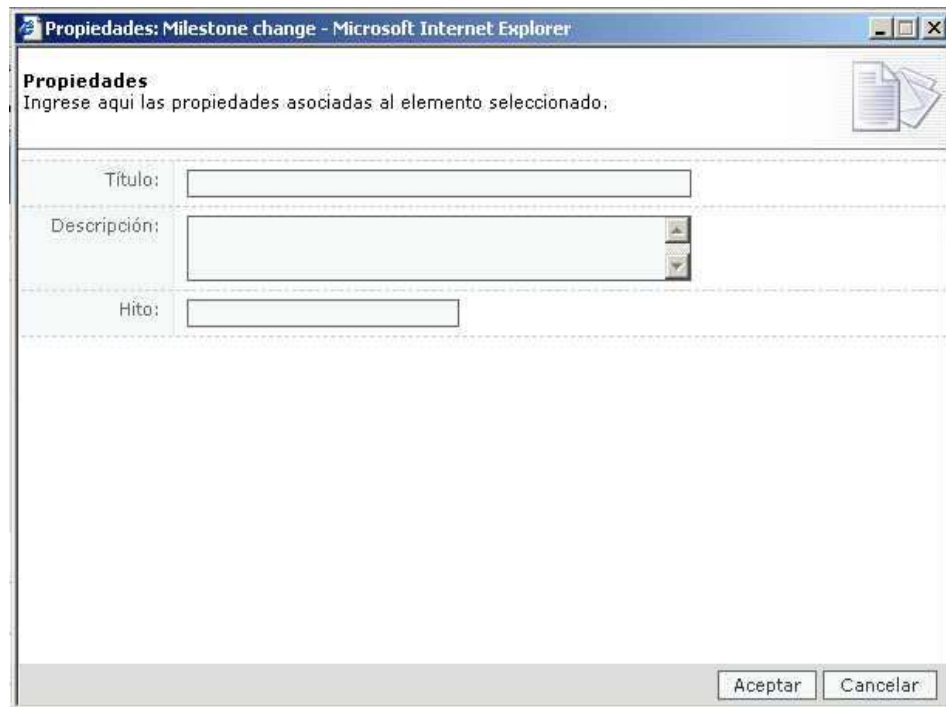


Figura 30

Como se puede apreciar en la figura anterior se cuenta con los siguientes campos:

- Un campo llamado "Título" en donde se ingresa el texto con el que va a aparecer esta actividad en el "área de diseño del flujo".
 - Un campo en donde si se desea se puede escribir una breve descripción sobre la actividad de condición.
 - Un campo llamado Hito en donde se ingresa cual es el Hito.
- Actividad de tipo "FIN": Todo proceso debe tener una actividad de este tipo, cuando el flujo llega una actividad de tipo "Fin" el proceso queda completado.

Ahora que ya se tiene una idea de como es el mecanismo para crear las distintas actividades y a partir de estas diseñar el flujo de un proceso, se va a estudiar como utilizar el editor de flujos para crear el flujo del proceso de solicitud de vacaciones.

Lo primero que se debe hacer es insertar la actividad de inicio en el "área de diseño del flujo", recordar que todo proceso debe tener una actividad de inicio y un esquema de información. En este caso como no se van a utilizar variables globales al proceso, en el formulario asociado a la actividad de inicio no se agregaran parámetros.

Una vez ingresada la información asociada a la actividad de inicio, el editor de flujos queda con la actividad agregada como se puede ver en la Figura 31.

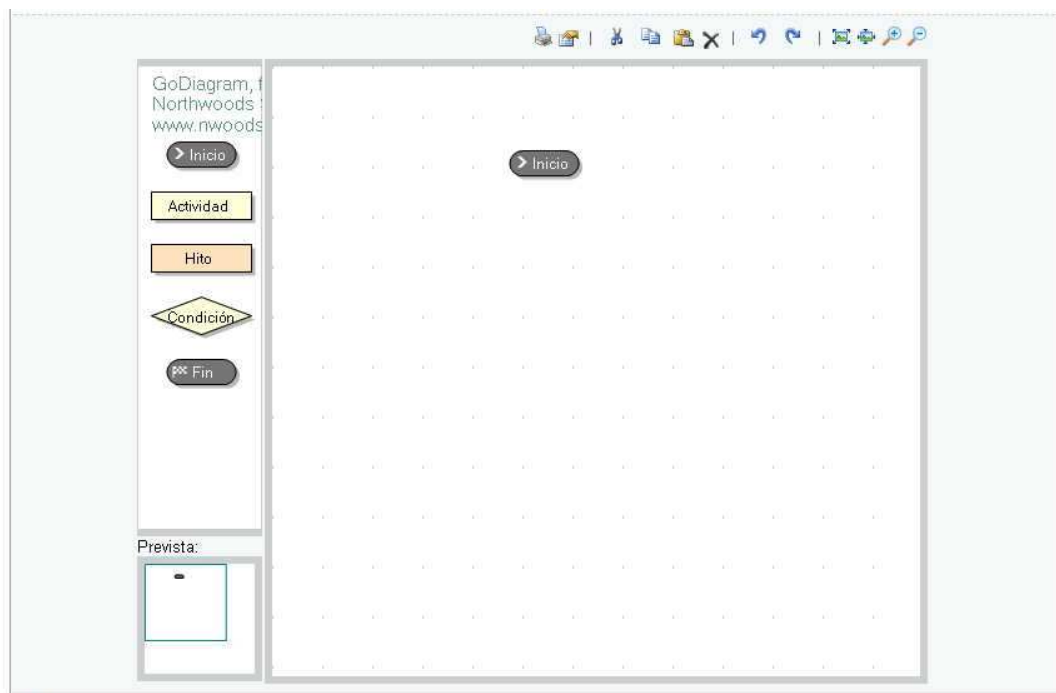


Figura 31

Una vez creada la actividad de inicio se agregará un hito para marcar que la solicitud de vacaciones del empleado ya fue ingresada. Para agregar este hito, se selecciona la actividad de tipo "Hito" de la parte izquierda del editor y se agrega en el "área de diseños del flujo", en ese momento se abre un formulario para completar los datos correspondientes al hito que en este caso son como se muestran en la Figura 32.

Propiedades Ingrese aquí las propiedades asociadas al elemento seleccionado.	
Título:	Solicitud Ingresada
Descripción:	La solicitud fue ingresada.
Hito:	Ingresado, a Aprobar.
Aceptar Cancelar	

Figura 32.

Lo que va a indicar el hito anterior es que se ingresó la solicitud por parte del empleado. Luego de completar los datos del hito aparecerá el mismo sobre el "área de diseño del flujo". Luego se crea una transición desde la actividad de inicio hasta este hito como se indica en la Figura 33.

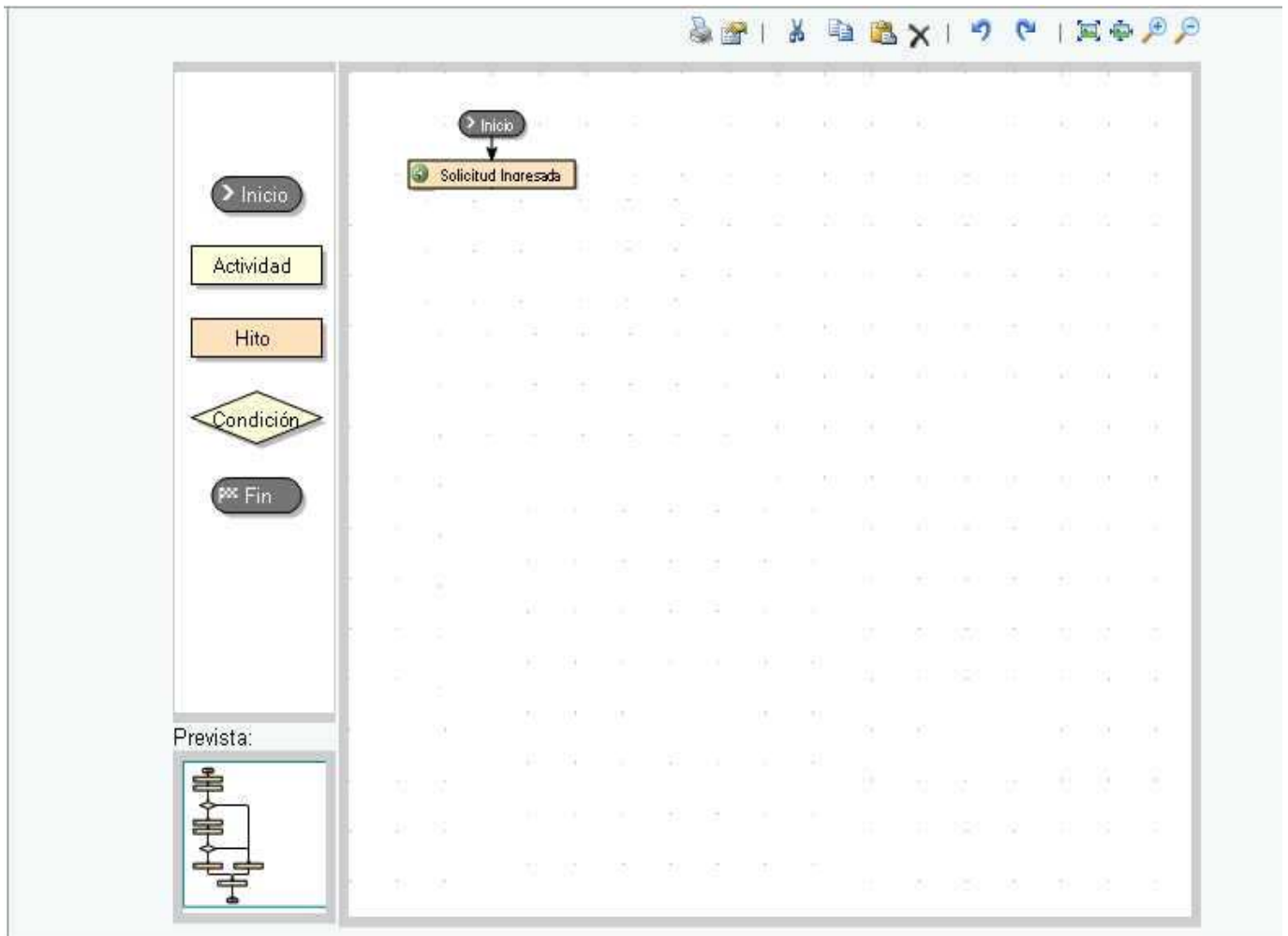


Figura 33.

La próxima actividad a agregar es aquella en donde el primer jefe debe o no, aprobar no la solicitud efectuada por el empleado. Esta actividad es del tipo "Participant Task" ya que tiene asociado un participante del workflow que es el responsable de realizarla. En este caso el responsable de la actividad debe ser el primer jefe y el formulario que debe completar se genera a partir del esquema que se ingresará en los datos de la actividad.

Para agregar esta actividad lo que se debe hacer es seleccionar de la parte izquierda del editor, el tipo "Actividad" e insertarlo en el "área de diseño del flujo", en ese momento se va a abrir el formulario asociado a este tipo de actividades para que se elija que actividad se desea crear. En este caso se debe seleccionar la actividad "Participant Task" como se muestra en la figura 34.

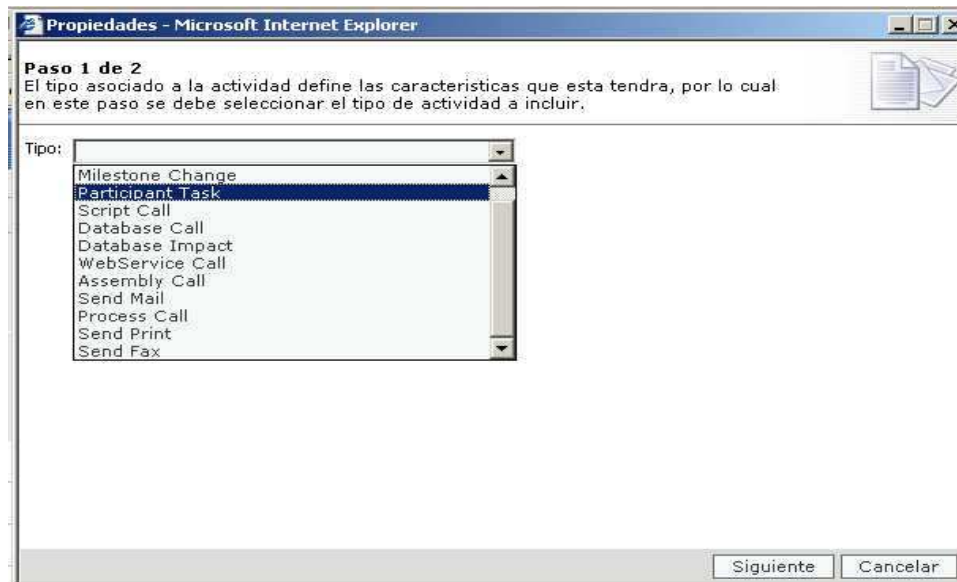


Figura 34

Una vez hecha la selección se presiona sobre el botón de siguiente y como se muestra en la Figura 35, se despliega el formulario con la descripción sobre las actividades "Participant Task".

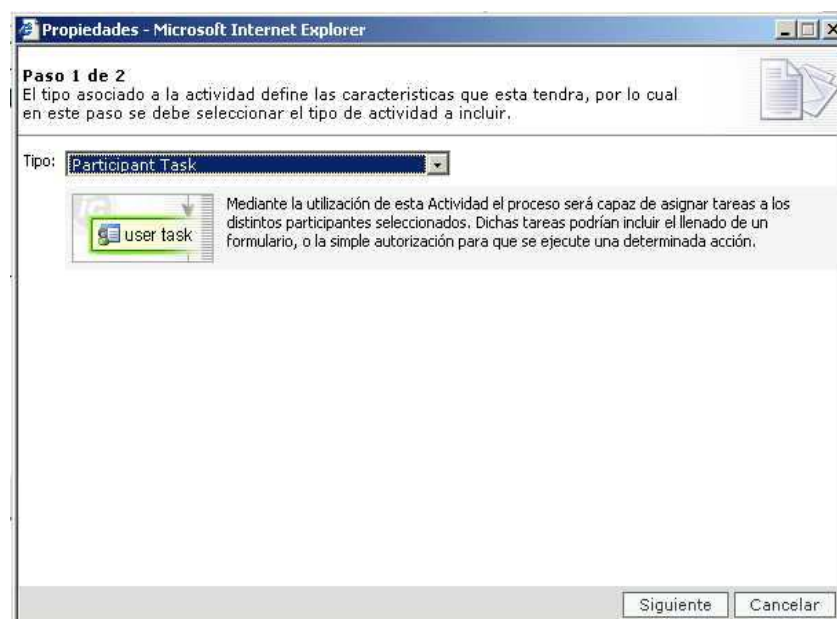


Figura 35

Luego se presiona el botón siguiente y se abre un nuevo formulario, en donde se deben completar todos los datos de la actividad, en este caso, la información correspondiente a un "Participant Task", como se muestra en la Figura 36.

Propiedades: Participant Task - Microsoft Internet Explorer

Propiedades
Ingrese aquí las propiedades asociadas al elemento seleccionado.

Título:

Descripción:

Prioridad de actividad:

Esquema/Formulario (opcional):

Límite: hrs.

Prioridad para participantes:

☒ Personal

☒ Programadores

☒ Avanzados

☐ Basicos

Participantes Seleccionados

Rodrigo Recoba

Participantes Disponibles

Figura 36

Como se puede apreciar el formulario anterior se divide en dos partes, en la parte superior se ingresan datos de la actividad como título, descripción, etc., mientras que en la parte inferior hay un navegador que permita seleccionar el o los participantes responsables por completar la actividad.

Para esta actividad los datos a ingresar son los siguientes:

- En el campo llamado "Título" donde se ingresa el título de la actividad, se ingresa "Aprobación". Este título es el texto que luego se va a ver sobre la figura que representa esta actividad en el editor de flujos.
- En el campo llamado "Descripción", se ingresa el texto "Formulario por el cual el usuario responsable aprueba o no la solicitud"
- Se ingresa que la prioridad de la actividad sea baja.
- Se selecciona que el esquema asociado a la actividad sea el llamado "Solicitud Licencia - Aprobacion1". Este esquema fue construido por el editor de esquemas y es a partir del cual se genera el formulario para que el participante responsable lo complete.

- En el campo llamado "Limite" se ingresa el valor 100, este valor son las horas que el proceso espera porque se complete la actividad, si una vez cumplido este tiempo el formulario asociado a la actividad no se completa el proceso se aborta.
- Se ingresa que la prioridad para los participantes sea baja.

Falta seleccionar el o los participantes responsables de completar la actividad. Para esto se cuenta con un navegador como se muestra en la Figura 37.



Figura 37

A medida que con el navegador se va "navegando" entre los distintos grupos y sub-grupos, en el área de texto llamada "Participantes Disponibles" que se encuentra en la parte derecha, se van mostrando los participantes pertenecientes al grupo o sub-grupo seleccionado.

En este caso se selecciona el participante "Rodrigo Recoba" que pertenece al sub-grupo avanzado dentro del grupo programadores como se puede ver en la Figura 38.



Figura 38

De esta forma se ha seleccionado que el participante "Rodrigo Recoba", sea el responsable de completar la actividad.

Se debe crear una transición entre el hito "Solicitud Ingresada" y la actividad Aprobación, para que el flujo pase del hito a dicha actividad como se puede ver en la Figura 39:

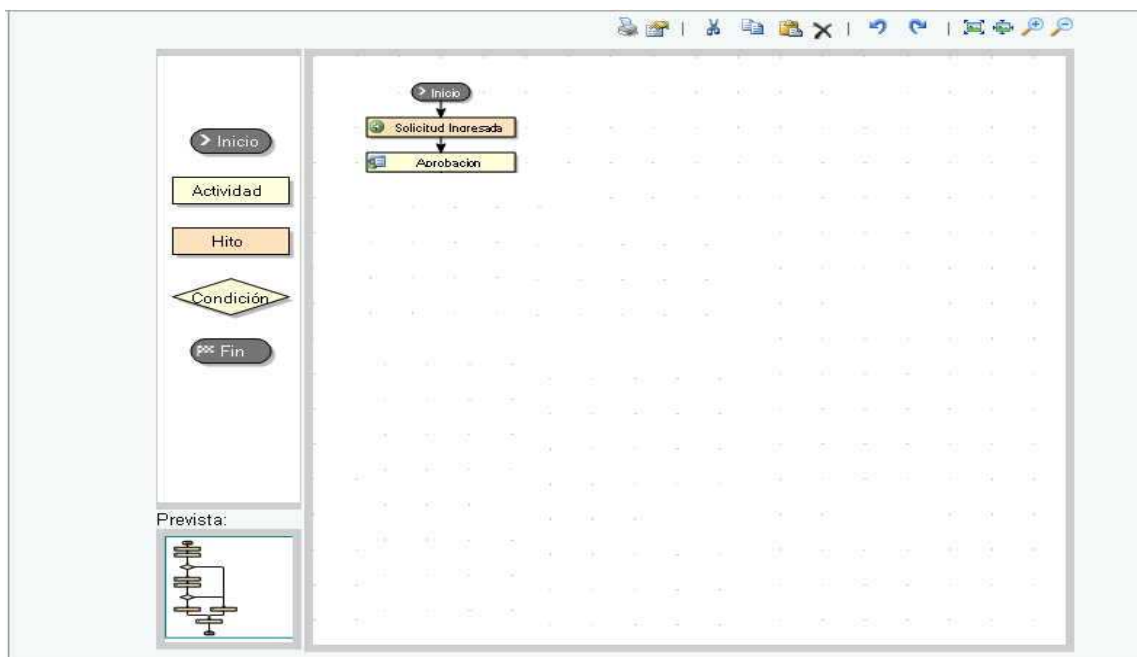


Figura 39

Lo que se debe hacer ahora es insertar la actividad de decisión que va evalúa la decisión del primer jefe, en este caso "Rodrigo Recoba", para esto se debe insertar la actividad de tipo "Condición" y en el formulario asociado a la actividad que se despliega en el momento de insertar la actividad en el "área de creación del flujo", se debe ingresar la condición que evalúa si el primer jefe aprobó o no la solicitud como se muestra en la Figura 40.

Id	Campo	Operador	Valor	Y/O		
1	{StateData.XmlInput.//Aprobacion1}	EQUAL	true	AND	Modificar	Eliminar

Figura 40

Los datos ingresados en el formulario anterior son:

- Igual que en las demás actividades en el campo "Título" se ingresa el texto que va a aparecer en la actividad en el editor de flujos, en este caso se ingresa como título "Aprobada?".
- La expresión que se ingresa como condición es "{//StateData.XmlInput.//Aprobacion1} = true". Lo que significa el operador izquierdo {//StateData.XmlInput.//Aprobacion1} es que del formulario que completó el primer jefe se obtenga el campo llamado "Aprobacion1". Entonces el significado de la condición es que el primer jefe haya aprobado la solicitud. Para más información sobre esta clase de expresiones ver Apéndice - Lenguaje para referenciar información del proceso (Solución Detallada).

Una vez agregada la condición "Aprobada" sobre el "área de diseño del flujo", se crea una transición que vaya desde la actividad "Aprobación" hasta dicha condición. El flujo del proceso una vez agregada esta condición se indica en la Figura 41.

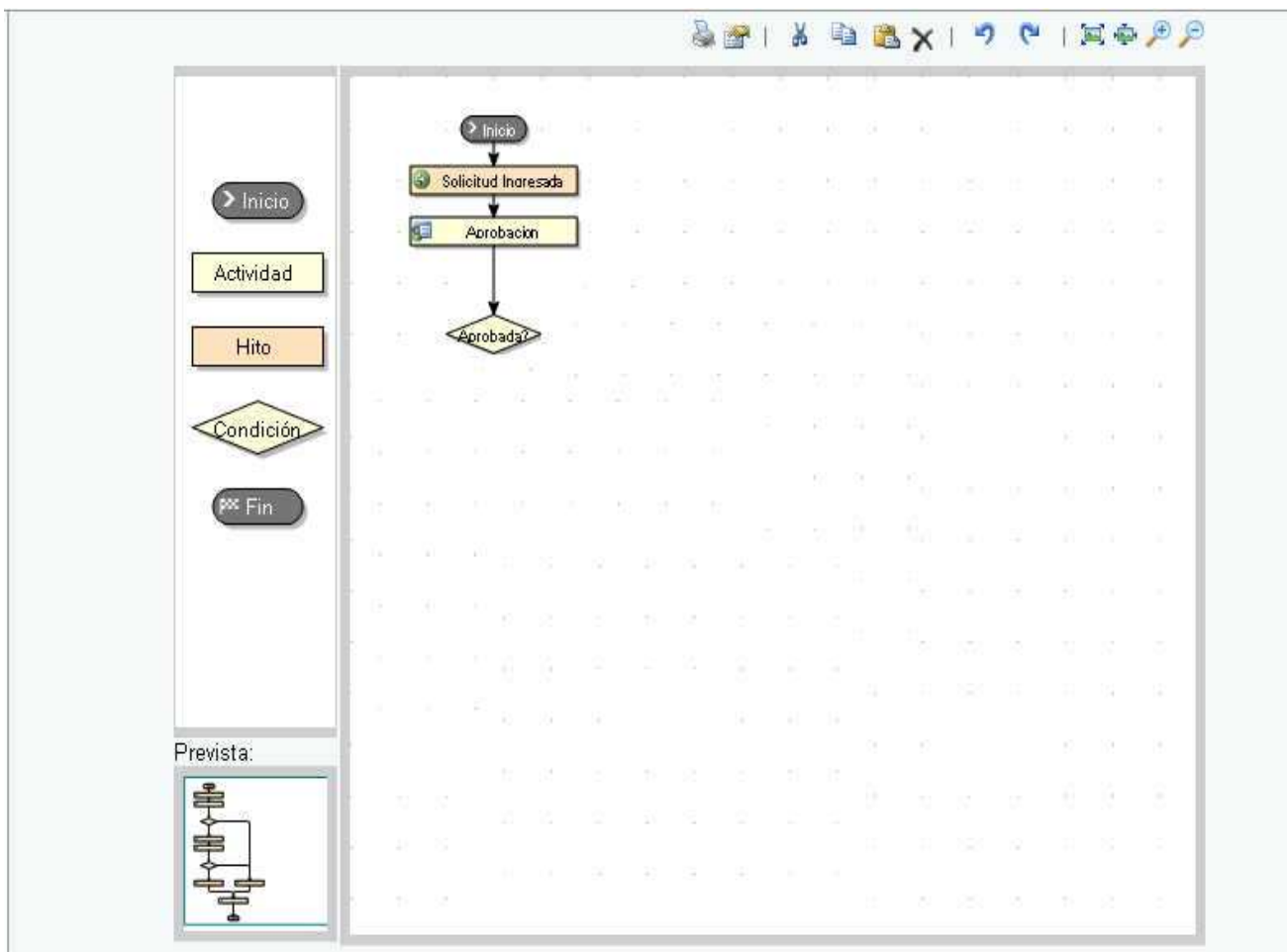


Figura 41

Luego se insertan 2 hitos más que indican cual fue el resultado de la evaluación. Un hito se llama "Solicitud Aprobada" y el otro "Solicitud Rechazada". Se crea una transición que va desde la condición "Aprobada" hasta el hito "Solicitud Aprobada" en caso de que el resultado de evaluar la condición "Aprobada" de verdadero y una transición desde la condición "Aprobada" hasta el hito "Solicitud Rechazada" en caso contrario.

El proceso una vez insertados los 2 hitos y las transiciones queda como se muestra en la Figura 42:

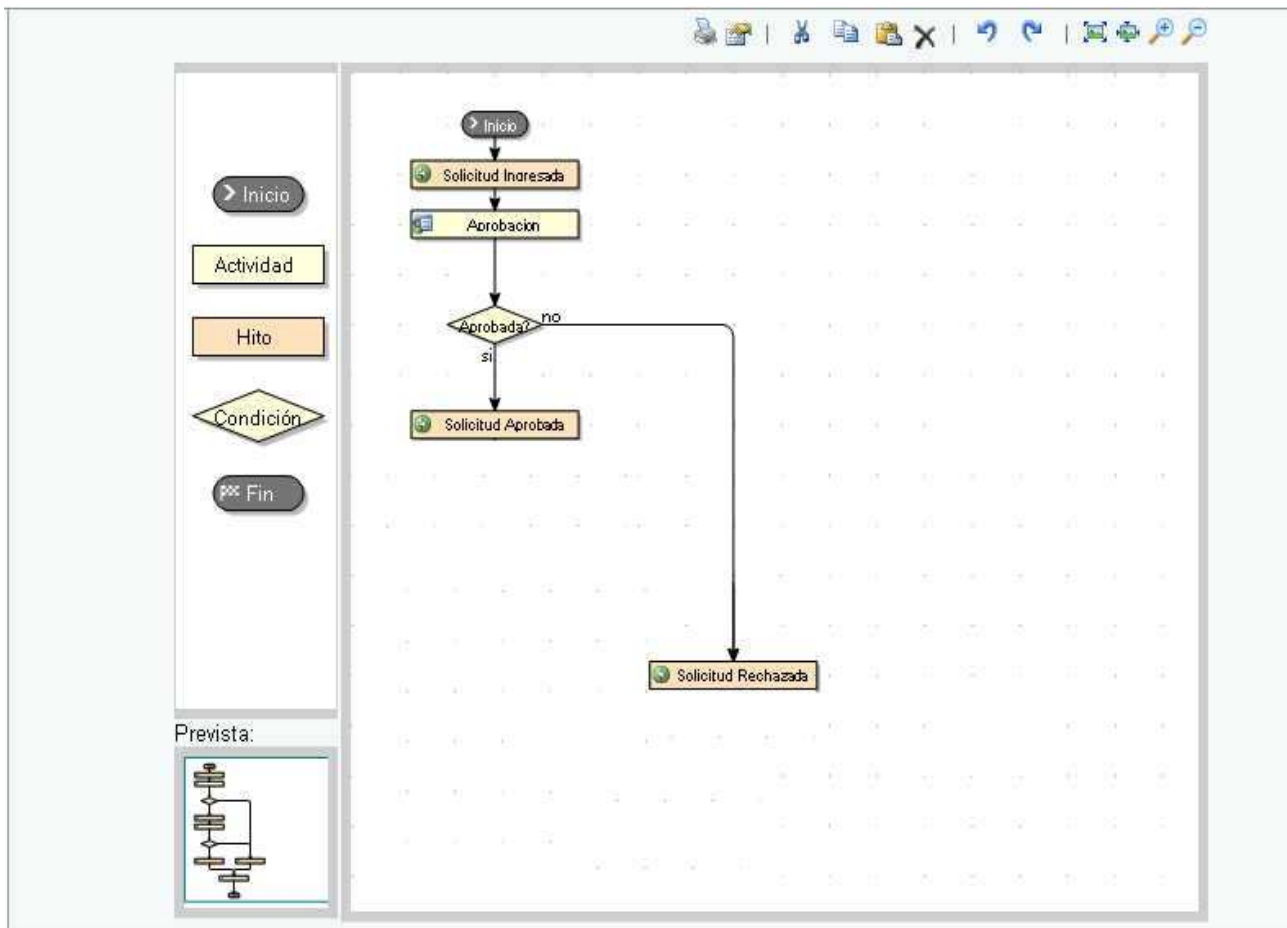


Figura 42

La próxima actividad a insertar es la actividad por la cual el segundo jefe aprueba o no la solicitud del empleado. Esta actividad al igual que la actividad "Aprobación", es de tipo "Participant Task" y el participante responsable va a ser el segundo jefe. El proceso para agregar esta actividad es similar al que se utilizó para insertar la actividad "Aprobación". El formulario se completa como se muestra en la Figura 43.

Propiedades: Participant Task - Microsoft Internet Explorer

Paso 2 de 2
Ingrese aquí las propiedades asociadas a la actividad.

Título: Verificacion

Descripción: Verificacion por un usuario responsable

Prioridad de actividad: Baja

Esquema/Formulario (opcional): Solicitud de Licencia - Segundo Supervisor

Límite: 100 hrs.

Prioridad para participantes: Baja

Participantes Notificaciones Parametros

☒ Personal

Participantes Seleccionados

Jose Casarini

Agregar Borrar Agregar Todos Borrar Todos

Participantes Disponibles

Programadores

Anterior Aceptar Cancelar

Figura 43

Los datos del formulario son:

- Como título de la actividad se ingresa "Verificación".
- La descripción es "Verificación por un usuario responsable".
- Se ingresa que la prioridad de actividad sea baja.
- Se selecciona como esquema asociado a la actividad el llamado "Solicitud de Licencia – Segundo Supervisor".
- El límite ingresado es de 100 horas.
- Se ingresa prioridad baja para los participantes.

En el navegador, se selecciona a "José Casarini" como participante responsable de la actividad. "José Casarini" es el responsable de verificar la solicitud del empleado.

Una vez insertada la actividad de "Verificación", se crea una transición que vaya desde el hito llamado "Solicitud Aprobada" hasta la actividad "Verificación" recién creada (Figura 44).

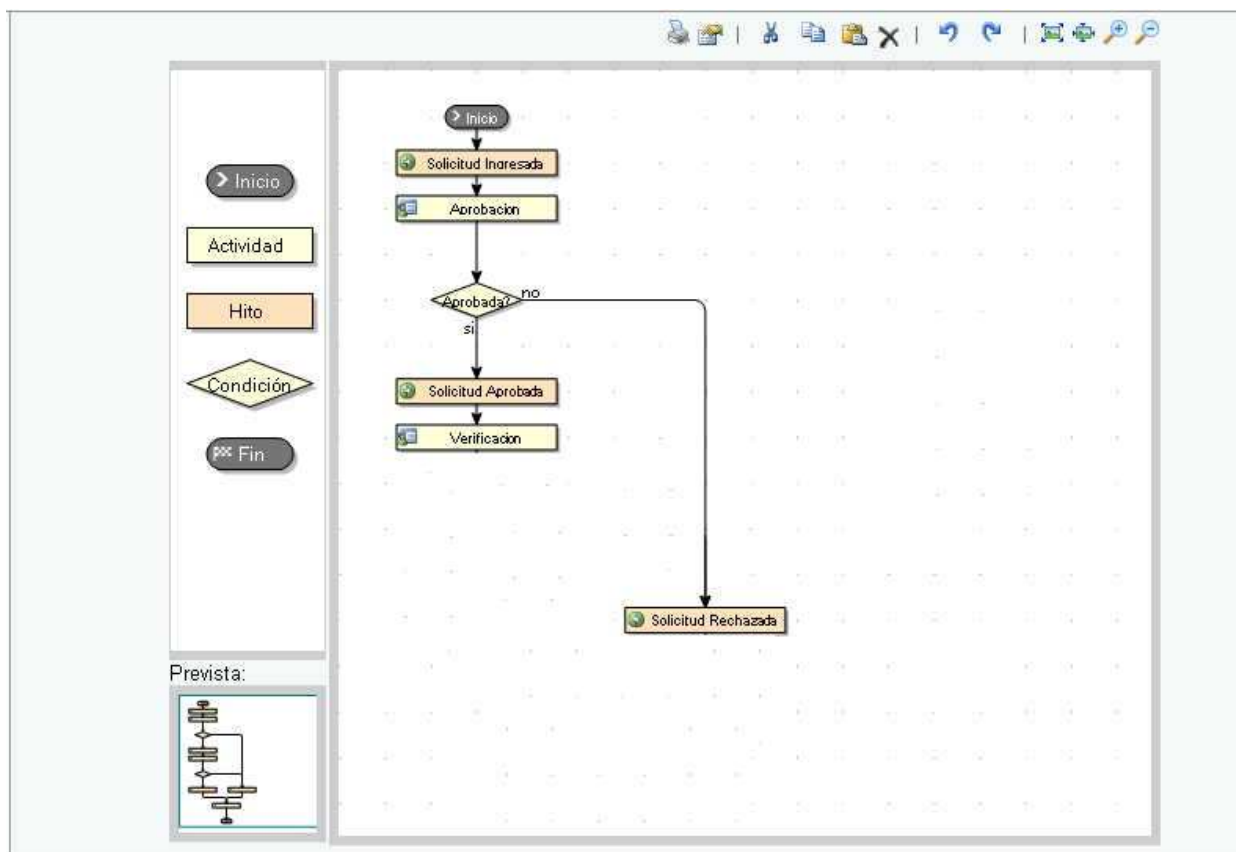


Figura 44

La próxima actividad a insertar es una actividad de condición que va a evaluar cual fue la decisión del segundo jefe. El formulario asociado a la condición es el siguiente:

Propiedades - Microsoft Internet Explorer

Propiedades
Ingrese aquí las propiedades asociadas al elemento seleccionado.

Título:

Descripción:

Condición:

Id	Campo	Operador	Valor	Y/O		
1	{StateData.XmlInput.//Aprobacion2}	EQUAL	true	AND	Modificar	Eliminar

Figura 45

Los datos ingresados el formulario anterior son los siguientes:

- Como título de la condición se ingresó "Verificada"
- La condición es la expresión: "{//StateData.XmlInput.//Aprobacion2} = true", que significa si el segundo jefe aprobó la solicitud del empleado.

Luego de agregada esta actividad el flujo es el indicado en la Figura 46.

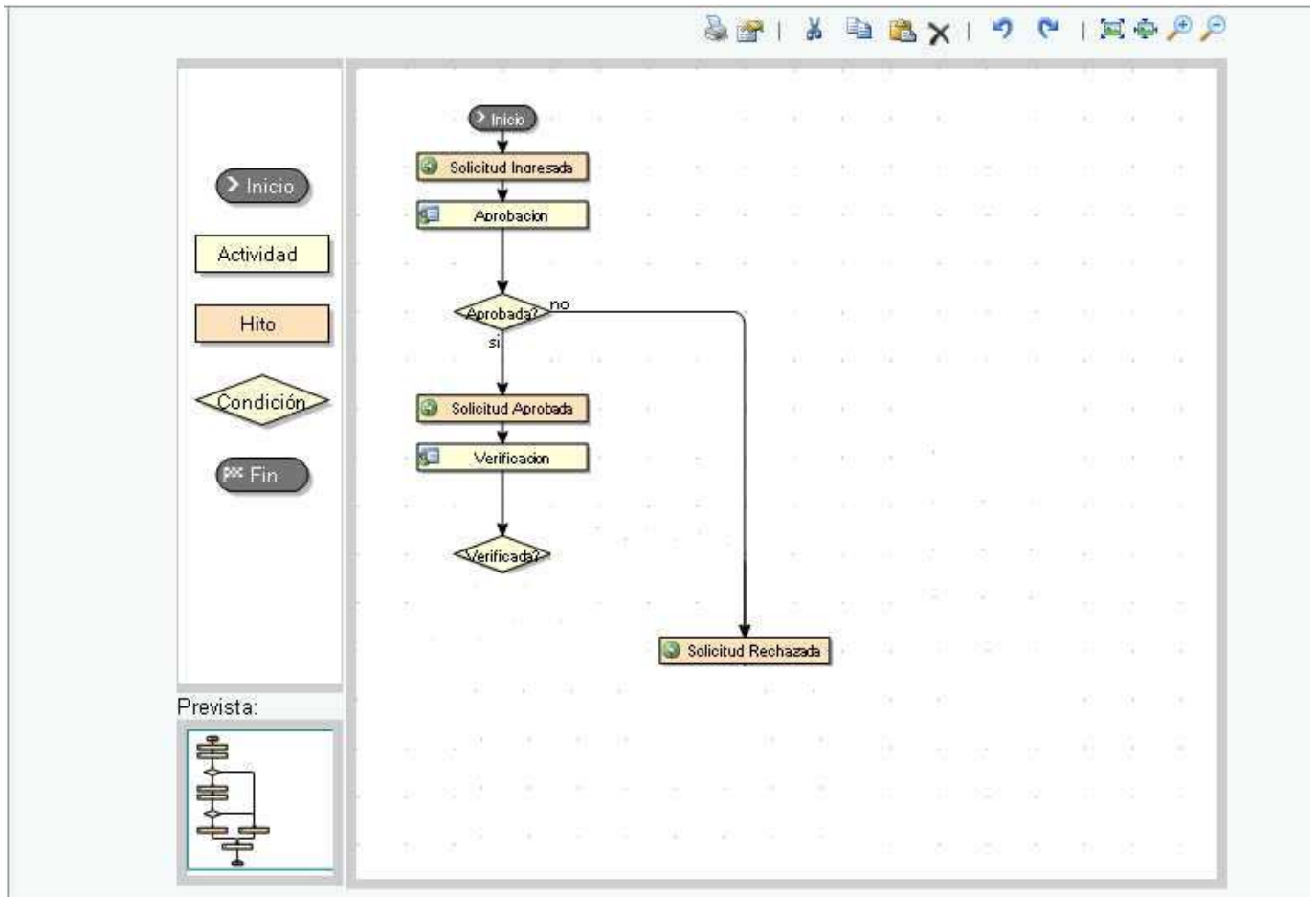


Figura 46

A continuación se agregan nuevo hito que marca cuando la solicitud fue aprobada. También se agrega una transición que vaya desde la condición "Verificada" hasta el Hito "Solicitud Aprobada" en caso de la evaluación de la condición sea verdadera y una transición desde la condición "Verificada" hasta el hito "Solicitud Rechazada" en caso de que la condición evalúe a falso. El estado del flujo es el indicado en la Figura 47.

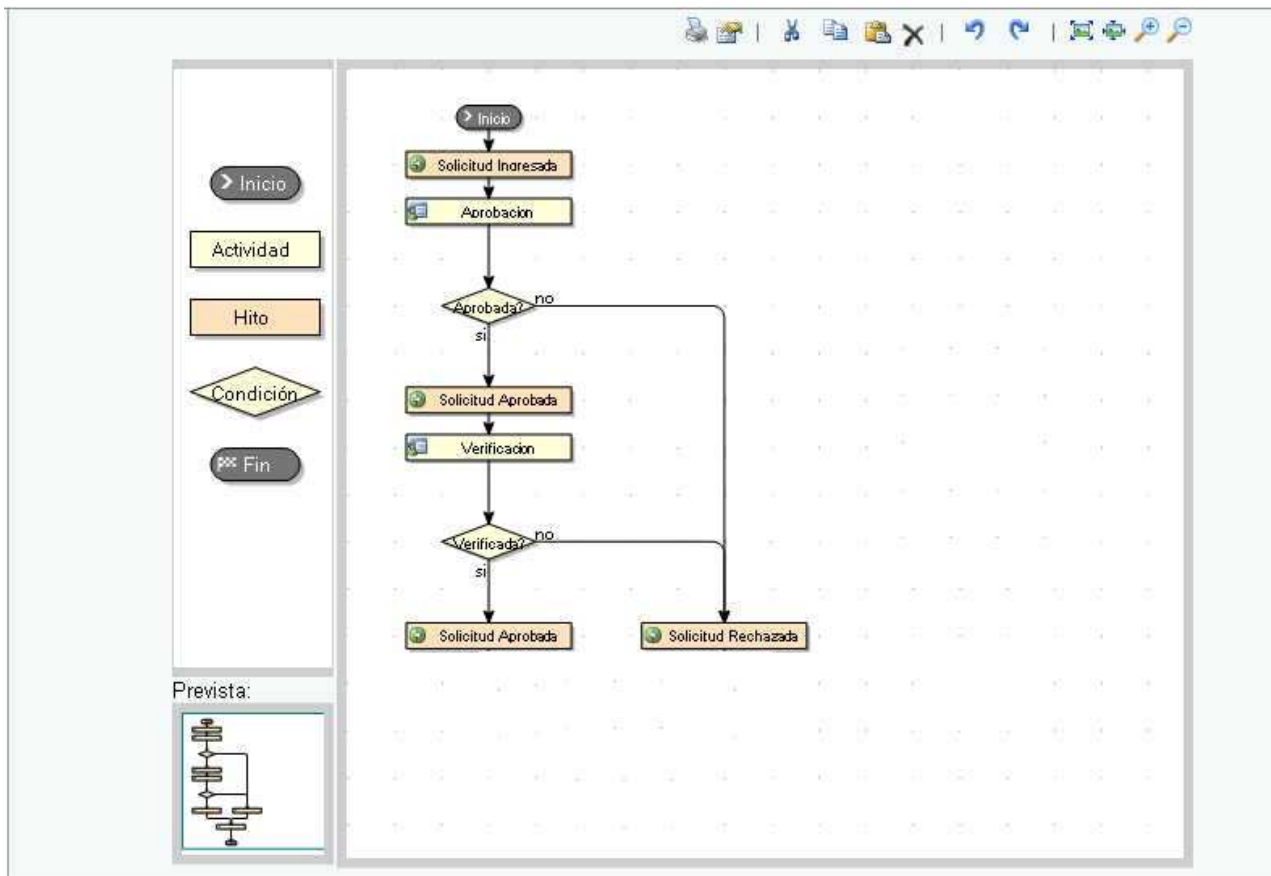


Figura 47

Queda ahora insertar la actividad que le notifica al usuario que hizo la solicitud cual fue el resultado de la misma. Para esto se va a insertar una actividad que envía mail. Igual a como se hizo anteriormente se inserta la actividad en el "área de diseño del flujo", se abrirá un formulario en donde se despliega un conjunto de actividades, se selecciona la actividad llamada "Send Mail", que es la que manda mails, se abre el formulario que muestra la descripción de ese tipo de actividades, se vuelve a presionar el botón siguiente y se abre el formulario para el ingreso de datos sobre la actividad que se completa como se muestra en la Figura 48.

Figura 48

Para el caso de la actividad "Send Mail" los datos se deben llenar de la siguiente forma:

- En el título de la actividad se ingresa "Notificación". Igual que en las demás estas actividades el título ingresado es lo que se va a desplegar sobre la actividad en el editor de flujos.
- Como descripción se ingresa el texto "Notifica al usuario el resultado de su solicitud".
- En el campo llamado "Servidor SMTP2", se debe ingresar un servidor de SMTP, que en este caso va a ser "Apolo".
- En el campo "De", se debe ingresar la dirección de mail desde el cual se va a mandar el correo con la notificación del resultado. En esta oportunidad la dirección de mail será "IC-Admin@Icorp.com.uy".
- Luego se debe ingresar el destinatario del mail. En este caso se quiere enviar el mail a la dirección que ingreso el empleado en el formulario de entrada al proceso, para esto en el campo llamado "Para", se ingresa la siguiente expresión "{StateData.XmlInput.//Mail}", dicha expresión lo que hace es tomar el campo llamado "Mail" del formulario que ingresó el empleado. Para más información sobre este tipo de expresiones ver el apéndice - Lenguaje para referenciar información del proceso.
- En este caso no se va a mandar ninguna copia del mail por lo que se dejan vacío el campo "CC" que es donde se ingresan las direcciones de mail que recibirán copia de este mail y el campo "CCO" que es donde se ingresan las direcciones de mail que recibirán copia oculta de este mail.

- Como asunto del mail se ingresa el texto "Solicitud de Licencia"
- Por último queda escribir en el campo "Mensaje" el texto del mail, que en este caso va a ser:
" Resultado Aprobación: {StateData.XmlInput.//Descripcion1}
Resultado Verificación: {StateData.XmlInput.//Descripcion2}".

Como se puede apreciar se utiliza la expresión "{StateData.XmlInput.//Descripcion1}" para obtener la descripción que ingresó el primer jefe en el formulario de la actividad "Aprobación", y la expresión "{StateData.XmlInput.//Descripcion2}" para obtener la descripción que ingresó el segundo jefe en la actividad "Verificación".

Luego de completar el formulario con todos los datos sobre la actividad "Send mail" se presiona el botón aceptar y aparece la actividad llamada "Notificación" sobre el "área de creación del flujo", se debe hacer una transición que vaya desde el hito "Solicitud Aprobada" hasta esta actividad de "Notificación", y otra transición que vaya desde el hito "Solicitud Rechazada", el flujo queda entonces como se indica en la Figura 49.

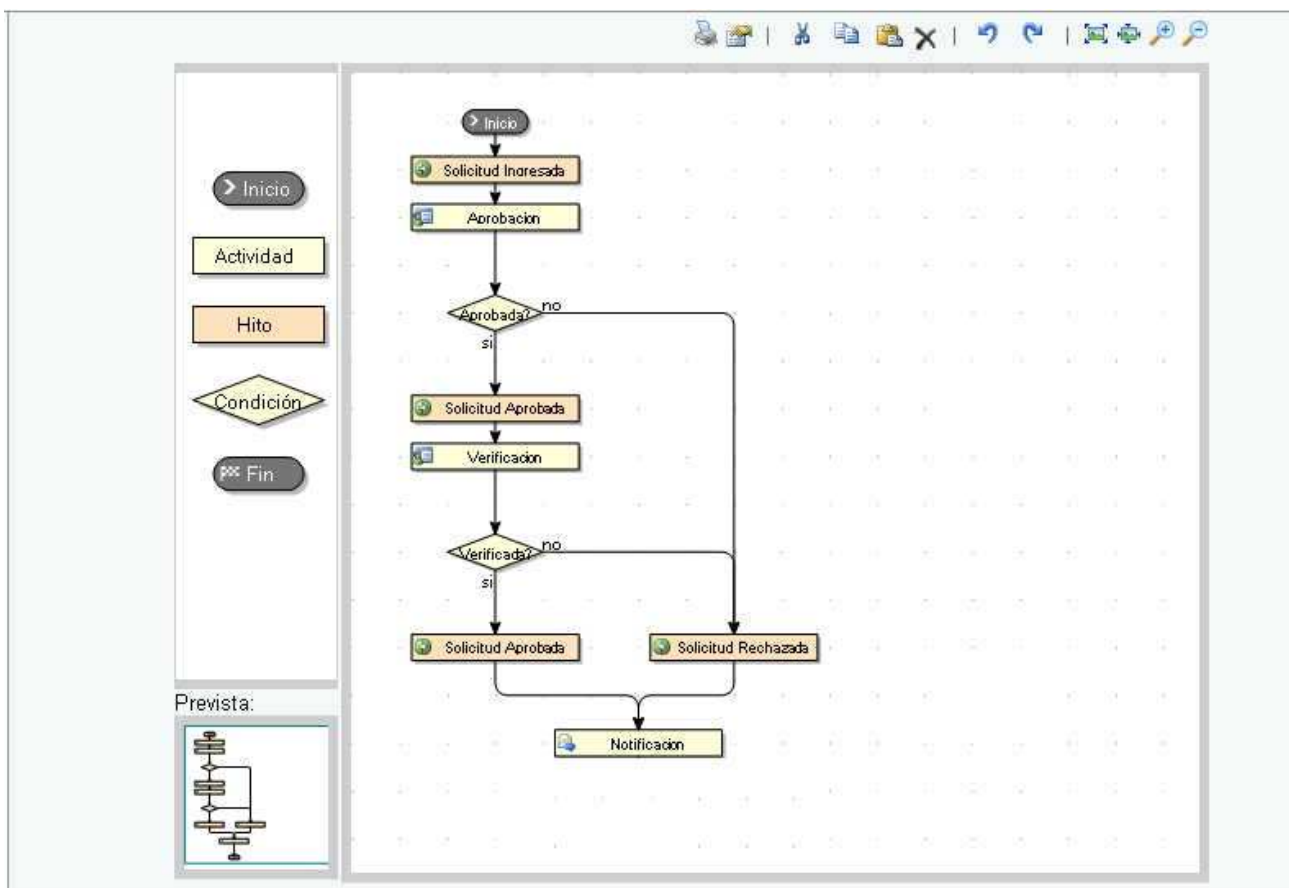


Figura 49

Lo último que resta por hacer es agregar la actividad final y crear una transición que vaya desde la actividad de "Notificación" hasta dicha actividad final. El flujo final queda como se muestra en Figura 50.

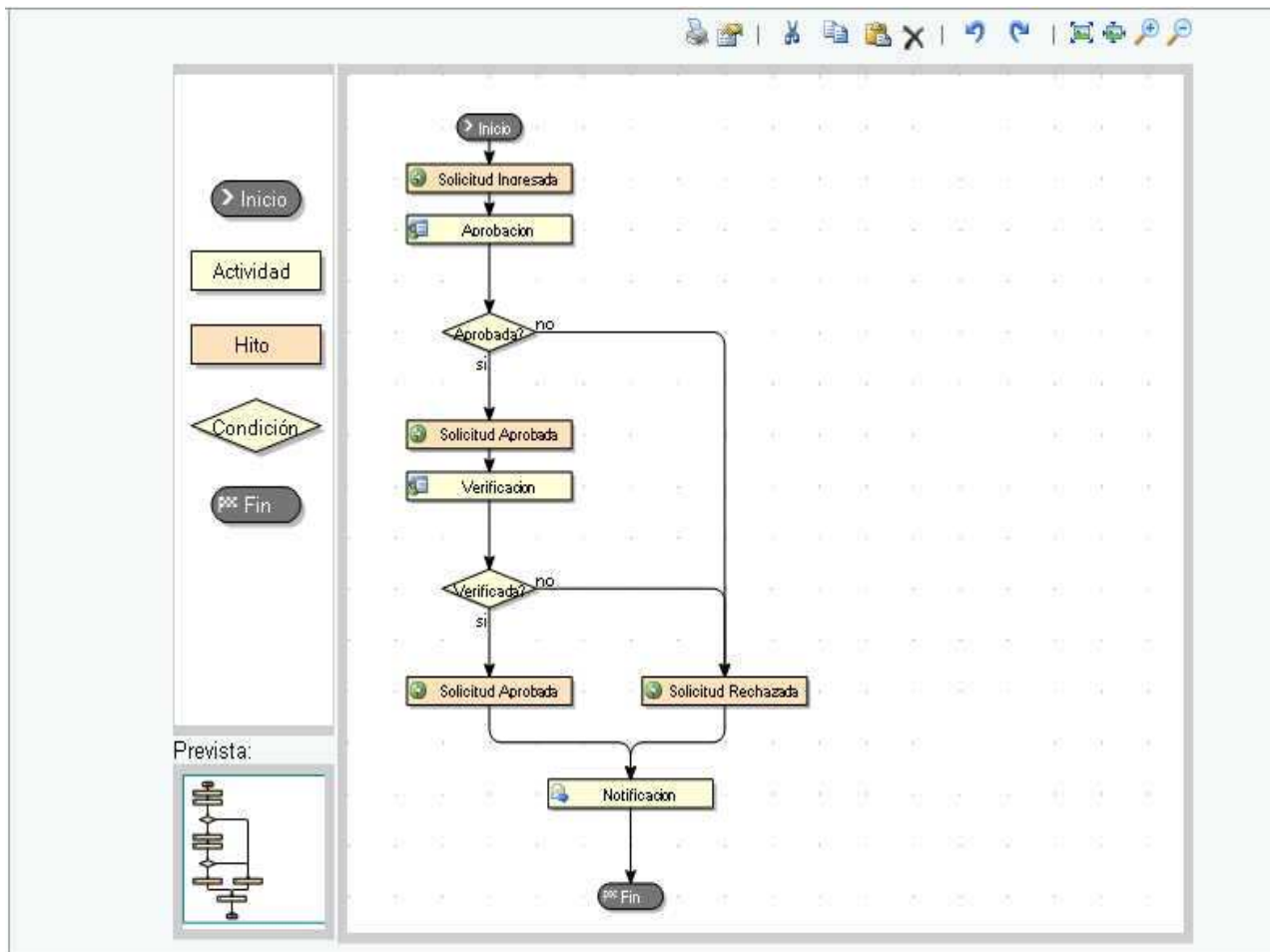


Figura 50

6.3 Transformación Final

En la parte inferior de la página de creación del flujo se encuentra el área en donde se puede ingresar una transformación XSLT como se puede ver en la Figura 46. De esta forma podemos transformar el xmlInput de entrada del proceso, para poder adaptarlo al esquema definido en el proceso, de esta forma se brinda mayor integración con otros sistemas.



Figura 51

Una vez diseñado el flujo se está en condiciones de ejecutar una nueva instancia del proceso. Se puede ver con claridad que el sistema de workflow permite es una plataforma de desarrollo de aplicaciones, ya que un cliente podría estar interesado adquirir un sistema de Solicitud de Licencia y no un sistema de workflow que permita diseñar todo el flujo. Si fuera así, alcanzaría con diseñar el flujo, implementar las actividades necesarias y las interfaces gráficas correspondientes para tener la aplicación funcionando.

7 TRABAJO A FUTURO.

Señalaremos en esta sección aquellos estudios que podrían realizarse en futuros proyectos, así como aquellos proyectos que la empresa realizará a la brevedad sobre el prototipo realizado.

7.1 Extensión del Prototipo.

Se realizará una extensión del prototipo que consistirá en:

- Extensión de los proceso a ejecutar
 - Se permitirá la ejecución de procesos que contengan ciclos, forks y joins, modificando el motor del workflow y el editor de formularios
- Integración de sistemas
 - Mediante la utilización de conectores se proveerán servicios de integración con sistemas propietarios y/o paquetes de terceros
- Integración de organizaciones
 - De forma similar a los conectores de integración de sistemas, se desarrollaran conectores/receptores orientados a la integración de organizaciones teniendo como característica principal, la disponibilidad, transaccionalidad y seguridad de las operaciones.
 - Alta disponibilidad mediante la utilización de prácticas de redundancia de servidores.
 - Soporte de certificados digitales para las comunicaciones inter-organizaciones.
- Automatización de procesos
 - Agente de Notificaciones: Un agente/servicio controlara los plazos máximos de ejecución de las tareas, enviando notificaciones a los involucrados y/o ejecutando procesos externos definidos en el propio flujo.
- Usabilidad
 - 100% Web, solo se requiere un Navegador de Internet, tanto para usuario Final como para un usuario Administrativo
 - Notificaciones mediante correo electrónico; en el contenido del correo se proveerá la información del proceso y tarea incluyendo links de acceso directo para las distintas acciones a ejecutar.
- Monitoreo y gestión de históricos
 - Consola de operaciones: Mediante una consola de operaciones se podrá visualizar la situación de los procesos activos, informando su estado, su responsable y consumos de recursos utilizado.
 - Reportes: Se proveerán variados reportes con los cuales se podrá acceder al histórico de procesos finalizados, visualizando datos básicos del mismo y todos aquellos datos sobre los cuales se hubiera indicado la necesidad de trazabilidad.
- Seguridad
 - Definición de Roles e integración con Active Directory para la asignación de privilegios.
 - Soporte de SSL para comunicaciones sobre Internet
 - Soporte para Certificados Digitales para integración de Organizaciones

7.2 Sistemas que se montarán sobre el sistema de workflow.

Infocorp pretende montar varios productos de su cartera sobre el sistema de workflow, como se indica en la Figura 14, implementando los procesos mediante la definición de flujos y personalizando las acciones particulares de cada sistema mediante componentes asociados a los flujos.

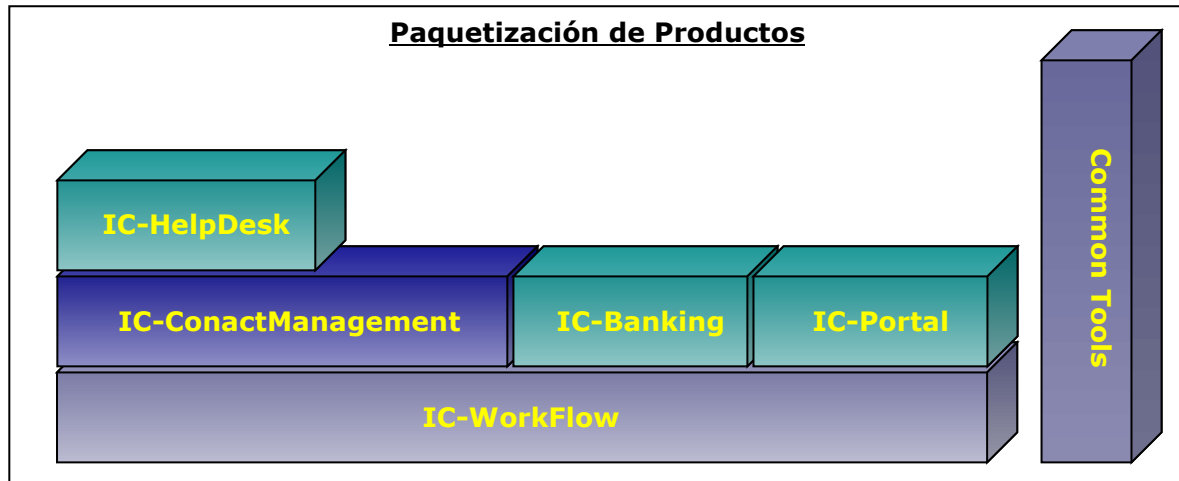


Figura 52

- IC-ContactMangement: Este producto presenta el concepto de "ficha de cliente" como elemento primordial, sobre el cual se definirán entidades relacionadas para la implementación de un escenario en particular. Ejemplo de uso:
 - IC-HelpDesk: Sistema para la gestión de servicio, reclamos incidentes de una organización.
 - IC-Negocios: Gestión de negocios y contactos asociados a un cliente.
- IC-Banking: Implementación de HomeBanking para la plaza financiera.
- IC-PortalWeb: Portal corporativo con alta capacidad de auto-administración de contenido mediante herramientas de generación de contenido orientadas al usuario.

8 CONCLUSIONES:

Cómo resultado del proyecto, se obtienen importantes nociones sobre la tecnología de workflow y se ponen en práctica métodos y conocimientos asimilados durante la carrera, para la posterior implementación de un prototipo que cumple con los requerimientos exigidos por Infocorp. Tanto el estudio, como el prototipo realizado, sirven como base para el sistema de workflow que agregará la empresa a su cartera de productos.

Se obtuvo un sistema de workflow que consiste en un motor que permite ejecutar flujos, un editor de flujos, un editor de esquemas, un generador automático de formularios a partir del esquema, un gestor genérico de participantes y un conjunto de aplicaciones que asisten la ejecución de flujos, realizando tareas como ser: envío de mails, ejecución de scripts, impactador en base de datos, ejecución de web services, ejecución de componentes realizandos en .NET.

El sistema de workflow realizado, además de ser un producto en si mismo, presenta las siguientes ventajas para Infocorp:

- Bajar costos de implementación y/o actualización de productos
 - Los ciclos de desarrollo de los proyectos se hacen más cortos gracias a la reutilización de código, librería, documentación.
 - Los costos de capacitación de recursos de desarrollo se verán disminuidas debido a que el producto como tal ofrecerá los servicios y/o caminos para implementar la lógica particular de un cliente.
- Productos con mayor calidad
 - El propio ICW obligara a que las personalizaciones se realicen en un entorno controlado, cumpliendo con estándares que mejoraran el resultado final.
 - Documentación reutilizable
- Actualización tecnológica
 - Tanto ICW como los paquetes a implementar se desarrollaran utilizando Microsoft .NET Framework sobre C# (tecnología de punta del mundo Microsoft).

Algunas conclusiones sobre la plataforma y el lenguaje utilizados:

- La gran cantidad de clases que brinda .NET fue de gran ayuda a la hora de la implementación del prototipo.
- Se facilita mucho el diseño y la creación de las interfaces web gracias al ambiente gráfico con el que se cuenta.
- Es muy simple y eficiente todo lo que refiere al manejo de bases de datos.

Como ya se mencionó antes el lenguaje utilizado para la creación del prototipo fue C#. La sintaxis de C# es muy parecida a la de Java, por lo que es muy fácil de aprender para programadores que estén familiarizados con Java. Es un lenguaje muy práctico, que esta pensado para ser utilizado sobre .NET.

El éxito del proyecto, promovió que Infocorp continuará su implementación, utilizando el prototipo realizado, los conocimientos adquiridos, y la documentación generada. Demostrando de esta forma su conformidad con el proyecto.

9 REFERENCIAS:

- [1] – David Hollingsworth - TC-1003 Workflow Reference Model – WfMC – 1995
<http://www.wfmc.org/standards/docs.htm>. - 20/5/2004
- [2] – Orfali, D. Harkey, J. Edwards - Cliente/Servidor Guia de supervivencia – Segunda Edición
Editorial Mc Graw Hill - 1998
- [3] – WfMC – TC-1011 Terminology & Glosary - 1999
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [4] – David Hollingsworth WfMC – TC-1023 Common Object Model - WfMC - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [5] – WfMC Work Group 1 – TC-1016-X Process Definition Interchange - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [6] – WfMC Work Group 1 – TC-1016-P Process Definition Interchange ,Process Model - 1999
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [7] – WfMC Work Group 1 – TC- 1016-O Process Definition Interchange – Organizational Model - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [8] – WfMC – TC-1025 Process Definition Interface – XML Process Definition Language - 2002
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [9] – WfMC – TC – 1009 Aplication Programming Interface (WAPI) – Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [10] – WfMC – TC- 1013 Aplication Programming Interface – Naming Conventions - 1997
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [11] – WfMC – TC – 1012 Interoperability – Abstract Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [12] – WfMC – TC -1015 Audit Data Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [13] – WfMC – Workflow Handbook 2003 - 2003
<http://www.wfmc.org/information/handbook03.htm> - 20/5/2004
- [14] – WfMC – Workflow Handbook 2002 - 2002
<http://www.wfmc.org/information/handbook02.htm> - 20/5/2004
- [15] – WfMC – Workflow Handbook 2001 - 2001
<http://www.wfmc.org/information/handbook01.htm> - 20/5/2004
- [16] – www.XData.com (XData)
- [17] – www.wfmc.org (Workflow Managment Coalition) - 20/5/2004
- [18] – www.urudata.com (QFlow) - 20/5/2004
- [19] - Tom Archer, Andrew Whitechapel - Inside C# - Editorial Microsoft Press - 2002

Referencias de interés

- www.bpmi.org (Bussiness Process Management initiative) - 20/5/2004
- www.msdn.microsoft.com/library/default.asp - 20/5/2004
 \Network and Directory Services\Network Communications\MessageQueue
 \Ado.Net
 \Asp.Net
- www.c-sharpcorner.com - 20/5/2004
- www.charphelp.com - 20/5/2004
- www.msdn.microsoft.com/vcsharp - 20/5/2004
- www.gotdotnet.com - 20/5/2004
- www.msdn.microsoft.com/netframework - 20/5/2004
- www.asp.net - 20/5/2004
- www.codeproject.com - 20/5/2004

PROYECTO DE GRADO – SISTEMAS DE WORKFLOW.

Javier Apaulaza, José Casarini, Rodrigo Recoba

Tutor: Guillermo Calderón
Usuario Responsable: Daniel Gómez

Instituto de Computación
Facultad de Ingeniería
Universidad de la República Oriental del Uruguay.

Marzo 2004

Resumen.

En este apéndice se presentan los estudios realizados sobre los sistemas de workflow. En el mismo se encuentran los conocimientos esenciales utilizados y aprendidos durante el proyecto. La mayoría de la información que contiene este documento fue extraída de documentos publicados por la WfMC – Workflow Management Coalition – por ser considerada como la agrupación de mayor influencia e iniciativa en el estudio de dichos sistemas.

ÍNDICE

1	ESTADO DEL ARTE.....	3
1.1	Orígenes y Evolución.....	3
1.2	Modelo de Referencia.....	8
1.3	Modelo de implementación.....	10
2	INTERFASE 1 - INTERCAMBIO DE DEFINICIÓN DE PROCESOS.....	17
2.1	Abstracto.....	17
2.2	Introducción.....	17
2.3	Intercambio de Definición de Procesos.....	18
2.4	Especificación brindada por la WfMC.....	19
3	INTERFASES 2 Y 3 – INTERFASE DE APLICACIONES DE WORKFLOW.....	31
3.1	Propósito.....	31
3.2	Perspectiva General.....	31
3.3	Definición de Interfases de Aplicaciones.....	31
3.4	Propósito.....	31
3.5	Descripción de las WAPI.....	32
3.6	Funciones WAPI de Conexión.....	32
3.7	Operaciones entre la API y el motor.....	32
3.8	Como opera una aplicación cuando usa invocaciones a la API.....	33
3.9	Funciones WAPI de control de procesos.....	33
3.10	Funciones WAPI de Control de Actividades.....	35
3.11	Funciones WAPI de Estado de Procesos.....	36
3.12	Funciones WAPI de Estado de Actividades.....	37
3.13	Funciones WAPI de lista de trabajo ("worklist").....	38
3.14	Funciones WAPI de Administración.....	40
3.15	Funciones WAPI de invocación de aplicaciones.....	41
4	INTERFASE 4 – INTEROPERABILIDAD.....	43
4.1	Definiciones.....	43
4.2	Estrategias de Interoperabilidad.....	45
4.3	Niveles de Interoperabilidad.....	47
4.4	Modelos de interoperabilidad.....	49
5	Referencias De Interes:.....	53

1 ESTADO DEL ARTE.

Se realizará un extenso estudio del estado del arte, sus comienzos, su evolución, estándares, tecnologías aplicadas, arquitecturas, diseños, terminología, brindando de esta forma un marco que permitirá comprender las diferentes decisiones tomadas, el estándar de referencia seleccionado, la arquitectura y diseño del Workflow implementado.

1.1 Orígenes y Evolución.

Si bien las personas son buenas para tomar decisiones, innovar, identificar hechos inesperados, no tenemos la misma capacidad para realizar tareas tales como: tener presentes cronogramas, fechas, documentos, realizar búsquedas, realizar seguimientos, verificar que los trabajos culminados pasen a su siguiente etapa, respetando una secuencia definida. Siendo que estas tareas son de vital importancia para las empresas es que surgió la necesidad de automatizarlas, surgiendo de esta forma el concepto de Workflow.

Al igual que la evolución de la informática en general, la evolución del Workflow está ligada con el cambio en los objetivos centrales de cada época. Si resumimos la evolución de la informática en las últimas cuatro décadas, podemos ver como han ido cambiando los objetivos a seguir en cada una de ellas. En la década de los 60' y 70' el gran objetivo era resolver grandes cantidades de calculo de manera eficiente. En los 80' se buscaba mejorar el manejo y administración de las bases de datos y en los 90' surge la necesidad de entender y manejar eficientemente los flujos de trabajo (Workflow), de manera de poder obtener el mayor provecho. Si miramos la actuación del Workflow dentro de estas tres etapas, podremos identificar lo que sería un Workflow Manual en la primera etapa, el Workflow Automatizado dentro de la segunda, y lo que ofrece un Workflow en la actualidad.

En el primer caso podemos ver que antes de que la informática se integrara al trabajo cotidiano, éste era realizado manualmente combinando toda la información en distintas carpetas. En este ambiente era bastante difícil determinar el estado de una determinada carpeta, así como también el hecho de determinar el proceso a seguir. Se manejaban grandes cantidades de documentos en forma manual, con los consiguientes errores humanos que traían aparejados dichos manejos. Por esto podemos identificar un Workflow Manual inmerso en las tareas cotidianas de esta época. Surge la necesidad de reemplazar las actividades manuales por actividades automáticas, es decir, se busca tener un mayor control y coordinación sobre toda la información que se maneja para llevar a cabo las tareas de las empresas.

Cuando entramos en la década del 80' se puede apreciar la existencia de diversos sistemas de información necesarios para llevar a cabo la producción de las empresas. Se ha logrado automatizar ciertas tareas que antes se realizaban manualmente, por esto podemos hablar de un Workflow Automatizado. A fines de esta década se busca mejorar el flujo de la información, el desafío que se plantea es obtener la información rápida y eficientemente. Surgen las necesidades de incrementar la eficiencia, optimizar la productividad, acortar los tiempos de procesos, tener un control sobre estos, así como también de reducir los costos y mejorar la gestión. Todo esto como consecuencia del incremento de la competitividad y de la exigencia de mejores productos, dentro de un mercado que avanza a gran velocidad.

Finalmente llegamos a la actualidad, donde nos encontramos con el objetivo de resolver eficientemente el Workflow. Actualmente existe una proliferación de diversos mecanismos de intercambio de información. Los mismos facilitan el manejo del flujo de información en general. Las metas son similares a las de épocas anteriores, pero el punto de partida, las asunciones, y el impacto son distintos. Dentro de la evolución actual del Workflow como tecnología identificamos la evolución y creación de ciertos productos que acompañan al mismo. Dichos productos son: Procesamiento de imágenes, Administración de documentos, Aplicaciones basadas en transacciones, Procesamiento de formularios, Correo electrónico, etc.

1.1.1 Introducción a los Sistemas de Workflow.

Son varias las definiciones que se han hecho del término "Workflow". A nuestro entender una de las definiciones más adecuadas es:

"Un Workflow es una estructura aplicada al movimiento de la información, para mejorar los resultados de los procesos de negocio por medio de la administración y coordinación de las actividades que desempeñan las personas involucradas en estos proceso de negocio."

[Workflow 94]

Independientemente de esta definición, existen tres conceptos claves dentro de lo que concierne a los workflows. Estos conceptos son actividad, coordinación y recursos. Un Workflow toma decisiones basadas en las condiciones del flujo determinadas durante el diseño, notifica a los recursos involucrados en el proceso, establece el trabajo que los mismos deben desarrollar, el estado en el que se encuentra actividad, etc..

Uno de los principales conceptos en los sistemas de Workflow son las personas. Para lograr el éxito en el desarrollo se debe conseguir que las personas realicen sus tareas y las notifiquen en tiempo y forma. Por ejemplo, una tarea en un Workflow podría ser notificar (vía mail) a una persona que debe realizar cierta actividad. Dicha actividad puede ser manual (por ejemplo colocar materia prima en una determinada maquinaria) por lo que su finalización, algunas veces, sería dada manualmente por el usuario. Si este no lo hace ni bien culmina la tarea, el Workflow podría estar detenido esperando por la finalización de la misma hasta que este así se lo comunique. Es por esto que el concepto persona es importante para la implantación de un sistema de Workflow, ya que para el éxito del mismo no alcanza con que se realice la tarea, sino que también se precisa que los usuarios lo acepten y adopten como parte del desarrollo de sus tareas diarias.

Los workflows facilitan la coordinación del trabajo entorno a las capacidades de compartir información. Por ejemplo, supongamos que estamos procesando una orden de compra, el Workflow conoce a quien (persona o proceso) debe enviar la misma para que la valide, verifique, etc. Es decir, el Workflow conoce cada una de las etapas por las que debe pasar un proceso, a quien asignarle cada tarea del mismo, y además es el encargado de automatizar el pasaje de una etapa a otra..

Como ya definimos un sistema de Workflow intenta automatizar total o parcialmente un proceso de negocio. Para esto, el mismo debe interactuar con diferentes procesos, en diferentes plataformas, ejecutados automática y manualmente, por maquinarias o humanos. La Figura 1 muestra un proceso de negocio conformado por diferentes procesos.

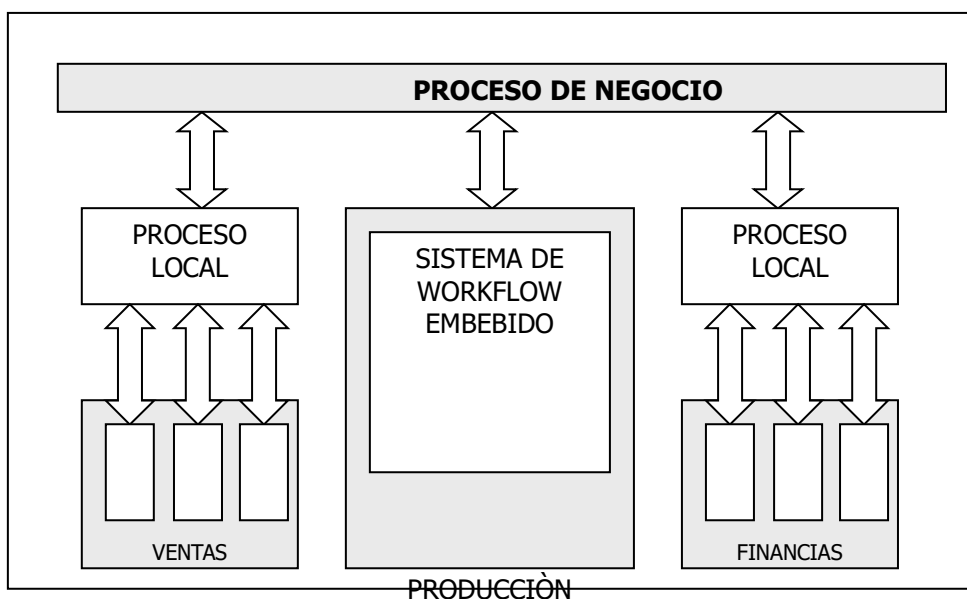


Figura 1

El Workflow será el encargado de ir ejecutando los diferentes procesos involucrados en un proceso de negocio, invocando diferentes sistemas existentes, otros workflow, etc. Es por esto, que el Workflow también oficia de integrador de los diferentes sistemas de la empresa.

Una solución de Workflow no implica una tecnología o producto particular. Existen variadas tecnologías, metodologías y herramientas que permiten construir sistemas automatizados, para estar seguros de escoger la mejor, se debe analizar cada aplicación en el contexto correspondiente.

Algunas de las características más importantes de los mismos, así como las principales funcionalidades que estos deben presentar.

- Se debe entender a la organización como un conjunto de personas que ejecutan y coordinan acciones, en donde cada persona demanda acciones de otra y la calidad de cada acción incide en el valor final. Un Workflow debe traducir el flujo del trabajo en la organización en un conjunto de procedimientos estructurados en donde el énfasis está dado por la coordinación de las acciones a medida que se ejecuta el trabajo.
- Cualquier individuo puede realizar una transacción en la medida en que esté autorizado para hacerlo. Es en ese momento en que la transacción fluye a través de la organización tomando diferentes rutas dependiendo de las acciones que se realizan.
- Un Workflow debe identificar responsables y tiempos de ejecución de cada etapa, proveer información sobre el estado de cada transacción, registrar y desplegar información histórica.
- Se deben soportar situaciones de paralelismo de actividades y de sincronismo de las mismas.
- Intercambio de información con los sistemas de información tradicionales.
- Mecanismos de seguridad.

1.1.2 Necesidad de Estandarizar

Se estima que actualmente los distintos productos de Workflow que hay en el mercado sobrepasan los cientos. Cada uno de ellos se enfoca sobre distintos aspectos funcionales, como ser, herramientas de diseño visual, en las cuales se ofrecen ciertos diagramas para representar la realidad. Otras se enfocan en la integración de los datos con las aplicaciones. El desarrollo de estándares para la interoperabilidad de las diversas herramientas, nos permitiría la elección de los mejores productos, según el enfoque que le demos a nuestra aplicación. Por ejemplo, podríamos comprar una herramienta que nos ofrezca un entorno amigable para realizar el análisis y diseño de nuestro problema, y por otro lado comprar una componente que nos resuelva la auditoria de los datos y poder trabajar en forma integrada con las dos componentes. Una de las

estrategias
denominadas
ser casos
situaciones
debe elegirse
claro caso

Vemos
un proceso

Es por lo
define

la es
puede
esta
esto
es es

es de

que

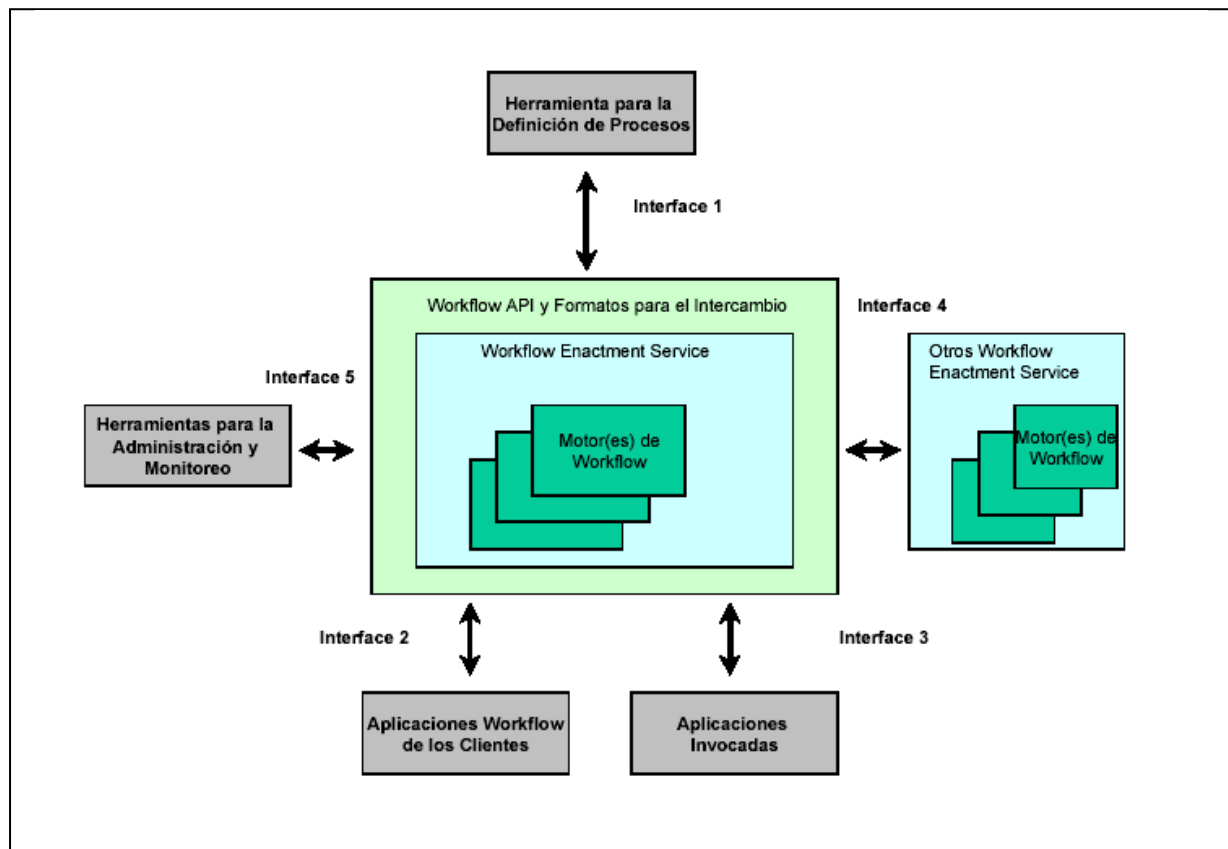


Figura 2

Explicaremos brevemente el significado de cada una de dichas interfaces, numerando algunos de los estándares que ya hay definidos.

Interfase 1: El propósito de esta interfaz es el intercambio de información que representa la definición de procesos, los cuales serán ejecutados por el motor del Workflow. Los estándares más conocidos para la definición de procesos son XPD (WfMC), BPQL(BPMI), XLang (Microsoft), WSFL(IBM).

Interfase 2: El propósito de esta interfaz es permitir el rehúso y la portabilidad de las aplicaciones. En esta interfaz se brindan APIs que permiten:

- Controlar Procesos y Actividades
- Manejo de worklists
- Manipulación de la definición de procesos

Estas WAPI s (Workflow API) fueron especificadas por la WfMC y OMG en 1998 en su segunda versión.

Interfase 3: Esta interfaz pretende brindar un framework para que terceros puedan integrar otras aplicaciones y servicios. Estas APIs son diseñadas para ser usadas por motor de Workflow o por el manejador de worklists. Permite invocar aplicaciones, pedir su estado, terminar, pausar, etc.

Interfase 4: El propósito de esta interfaz es permitir que los procesos de negocio sean implementados por dos o más sistemas de Workflow.

Interfase 5: Intenta permitir una administración consistente entre diversos sistemas. Esta interfaz brinda un conjunto de operaciones sobre las instancias de los procesos y actividades, consulta de estados y un amplio espectro de funciones administrativas

En los apéndices subsiguientes se describirá cada una de las interfaces y sus funcionalidades, clarificando de esta forma el porque de dicha división.

1.2 Modelo de Referencia

Se presentará a continuación modelo de referencia para WFMS (Workflow Management Systems), procurando proveer un framework para el desarrollo de sistemas de workflow.

Para clarificar las ideas que se desarrollaran a lo largo de este documento, introduciremos los conceptos de workflow y workflow management system. Un workflow puede definirse como "la automatización total o parcial de un proceso de negocio", por lo cual definimos al WFMS como "un sistema que define, maneja y ejecuta workflows cuyo orden de ejecución es dirigido por una representación de la lógica del workflow".

A alto nivel, el WFMS se puede caracterizar por proveer soporte a las tres áreas funcionales que describiremos a continuación.

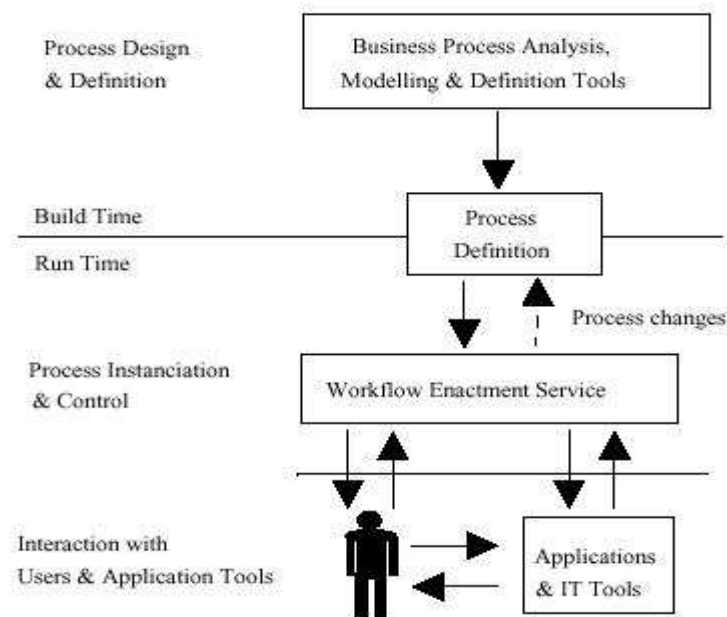


Figure 1- Workflow System Characteristics

- 1- Definición y Diseño del proceso: Donde se define y modela el proceso de workflow y las actividades que lo componen.
- 2- Control: Maneja los procesos del workflow en un ambiente operacional, secuenciando las diferentes actividades que forman parte de cada proceso.
- 3- Interacción con usuarios y aplicaciones: Donde se procesan los diferentes pasos de cada actividad.

La figura anterior nos muestra el workflow enactment service como el componente medular presentando interfaces con usuarios y aplicaciones distribuidas. Cada una de estas interfaces es un importante punto de integración entre el workflow enactment service y otros componentes o infraestructuras, por lo cual serán detalladas más adelante al igual que el workflow enactment service.

Una de las principales habilidades de un workflow es la distribución de tareas entre los diferentes participantes. La función de distribución debe operar a diferentes niveles dependiendo del alcance del workflow; este debe usar una gran variedad de mecanismos de comunicación (mail, objetos distribuidos, etc.). Una vista de la distribución de la arquitectura del software, puede ser la mostrada en la figura 2.

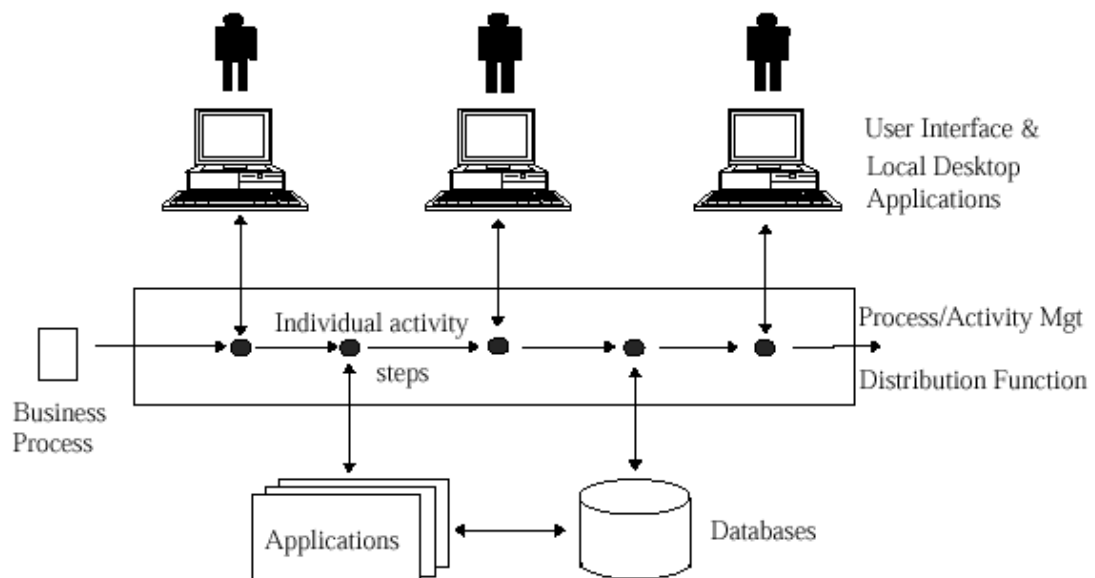


Fig 2 - Distribution within the workflow enactment service

1.3 Modelo de implementación.

A pesar de existir una gran variedad de productos en el mercado, es posible construir un modelo de implementación general de un WFMS que puede coincidir con la mayoría de los productos existentes, creando de esta manera una base común para el desarrollo de escenarios interoperables.

En esta aproximación identificamos los principales componentes e interfaces de un WFMS. Los principales componentes de un WFMS genérico son mostrados en la figura 3. Este modelo genérico muestra tres diferentes tipos de componentes.

- 1- Componentes de Software, los cuales proveen soporte para varias funcionalidades del WFMS (mostradas en negro)
- 2- Diferentes tipos de definición de sistemas y control de información (sin colorear)
- 3- Aplicaciones (coloreadas en gris) que no son parte del producto de workflow, pero pueden ser invocadas como parte del sistema de workflow.

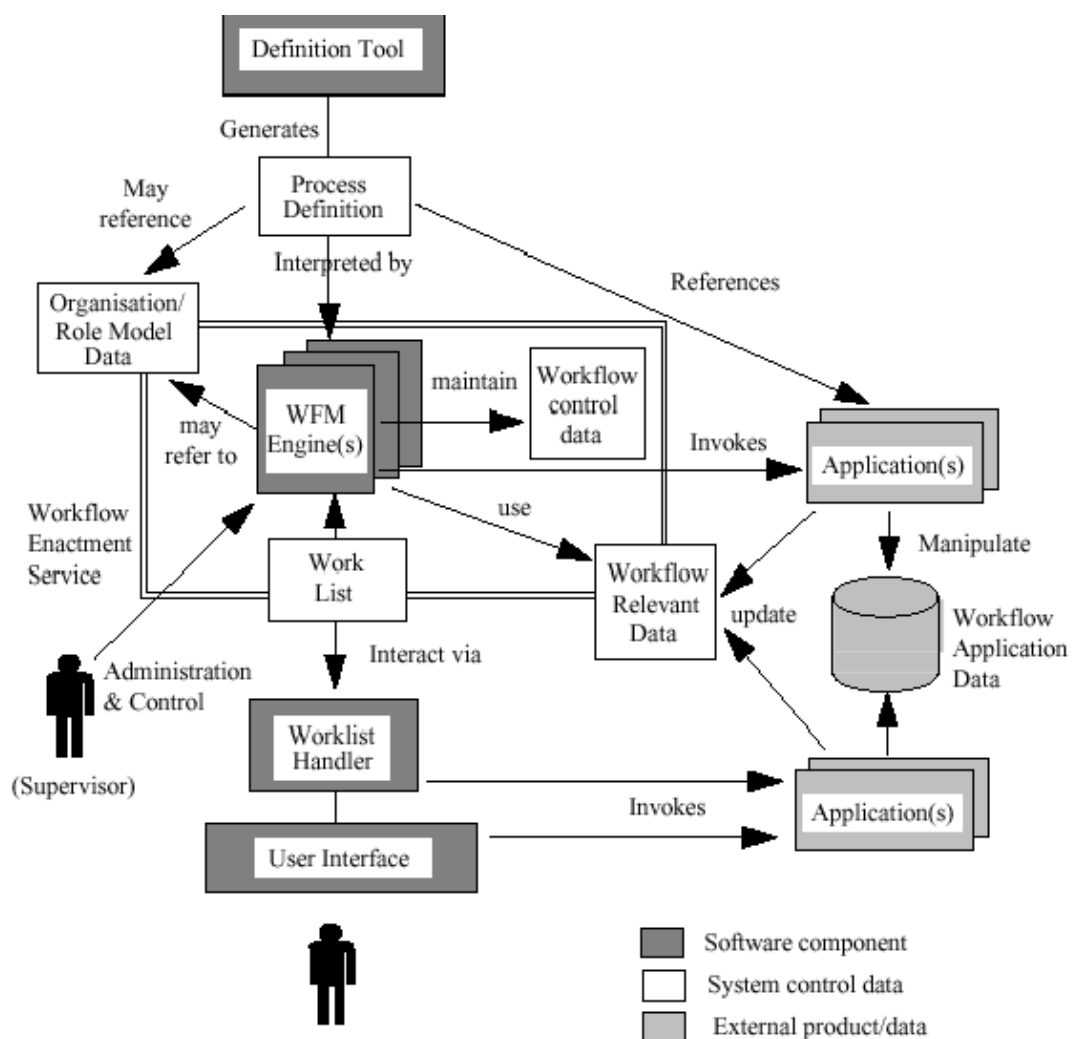


Figure 3 - Generic Workflow Product Structure

Nuevamente podemos ver la importancia del workflow enactment service, el cual interpreta la descripción del proceso y controla la instanciación de procesos, agregando work items a la lista de trabajo del usuario, además invoca herramientas a medida que es necesario. Estas tareas pueden ser realizadas mediante uno o más motores del workflow, los cuales manejan la ejecución de distintas instancias. El workflow enactment service mantiene control de la información de manera centralizada o distribuida mediante un conjunto de motores de workflow.

Existen, por lo tanto, diferentes alternativas para desarrollar el workflow enactment service, las cuales mostramos en la figura 4. Estos escenarios muestran que ambos servicios de workflow exhiben propiedades comunes en sus límites, pero difieren en su arquitectura interna. La idea principal es, entonces, definir de manera adecuada los límites del workflow enactment service, el cual debe exhibir funciones estándar a través de APIs.

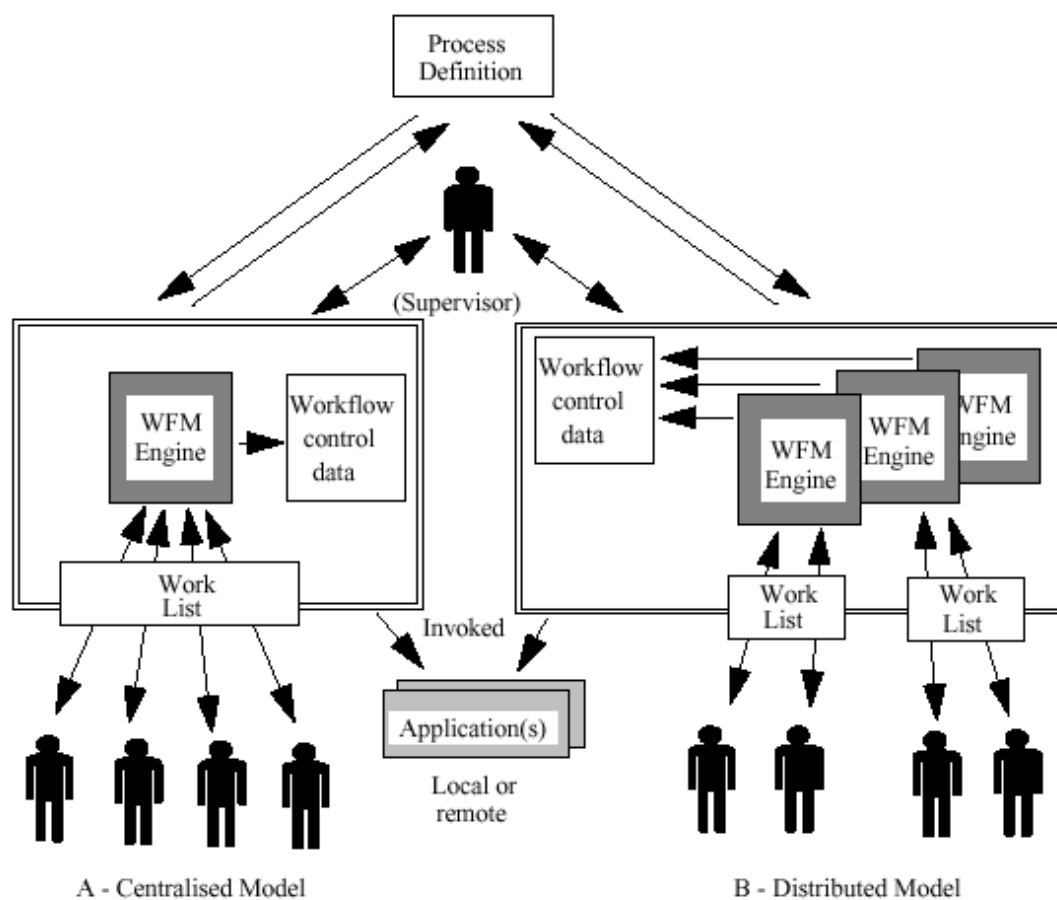


Figure 4 - Standard workflow enactment service boundary

Otro importante punto a decidir en la arquitectura de nuestro Workflow, es como desarrollaremos nuestras aplicaciones cliente. En el modelo la interacción ocurre entre el worklist handler y un motor del workflow a través de una interfaz bien definida. En el nivel más sencillo la worklist es accedida por el motor del workflow simplemente para asignar work ítems, y por el worklist handler con el propósito de presentar las tareas al usuario para su procesamiento.

Pero existen varias implementaciones posibles de realizar esta interacción, las cuales son mostradas en la siguiente figura.

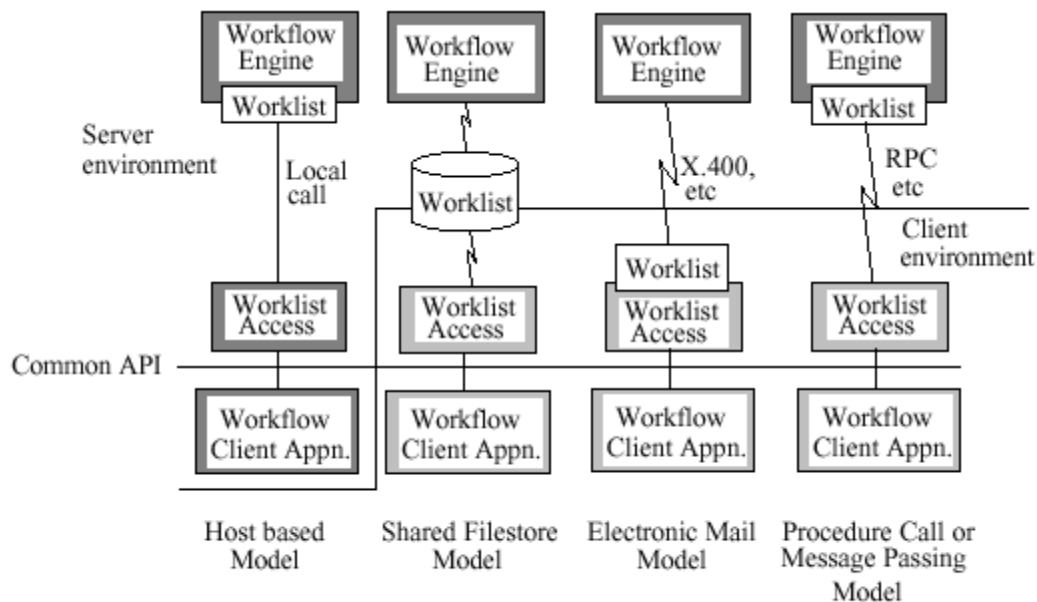


Figure 5 - Alternative client worklist handler implementations

Modelo de Workflow.

Este modelo fue realizado a partir de una estructura genérica de un workflow, identificando las interfaces de dicha estructura, de forma de permitir interoperar a los diferentes productos. Dicho modelo es mostrado en la figura 6. La interfaz del workflow enactment service es denominada WAPI – Workflow APIs e Intercambio, las cuales deben ser consideradas como un conjunto de constructores mediante los cuales los servicios del workflow pueden ser accedidos.

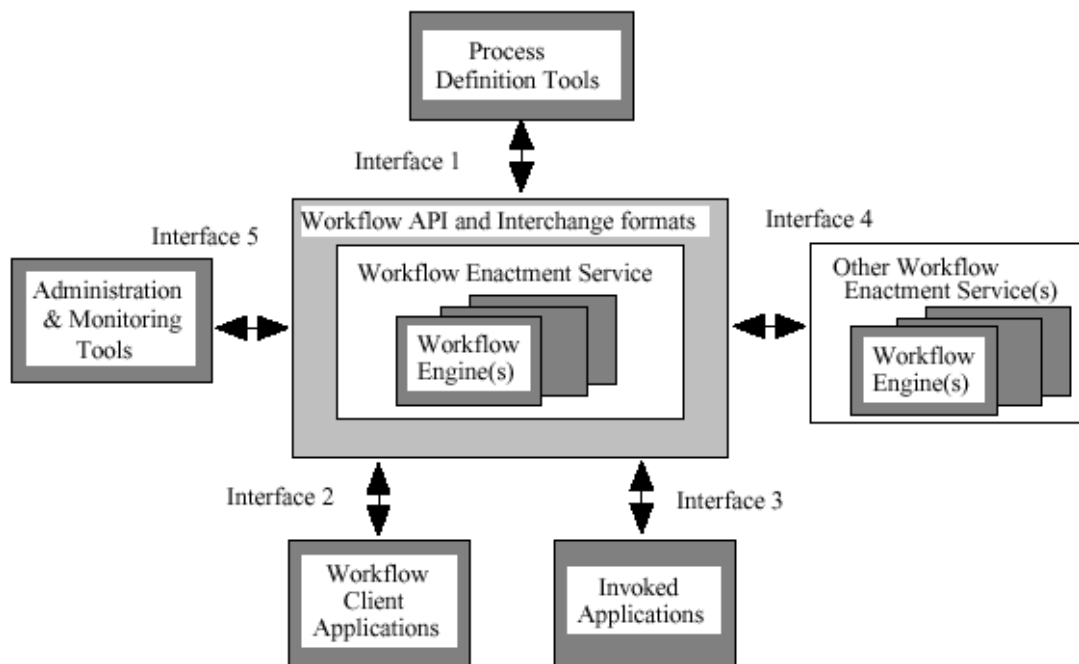


Fig 6 Workflow Reference Model - Components & Interfaces

Interfase 1: Es la interfaz entre las herramientas de modelo y definición de procesos y el WFMS. Esta interfaz debe soportar el cambio de definición de procesos.

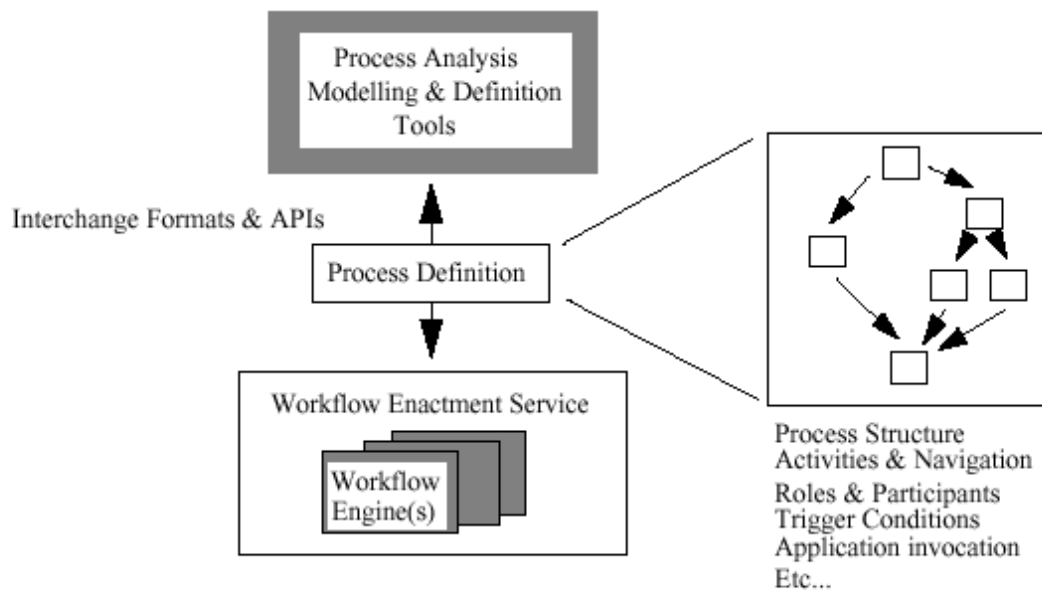


Fig 9 - Process Definition Interchange

La forma indicada en la figura 9 nos permite utilizar herramientas de modelado independiente, así como la posibilidad de exportar definiciones de procesos a diferentes productos de workflow.

Interfase 2: El manejador de la worklist es el componente de software que interactúa con el usuario final, en aquellas actividades que requieren recursos humanos. El worklist handler debe ser brindado como parte del WFMS o bien puede ser escrito por el usuario.

La internase entre la aplicación cliente y el motor del workflow debe ser lo suficientemente flexible en términos de:

- Identificación de procesos y actividades
- Direcciones y nombres de recursos
- Información y estructuras
- Mecanismos alternativos de comunicación.

Una aproximación a la API es mostrada en la figura 11, esta especificación será realizada en otro documento de forma más detallada.

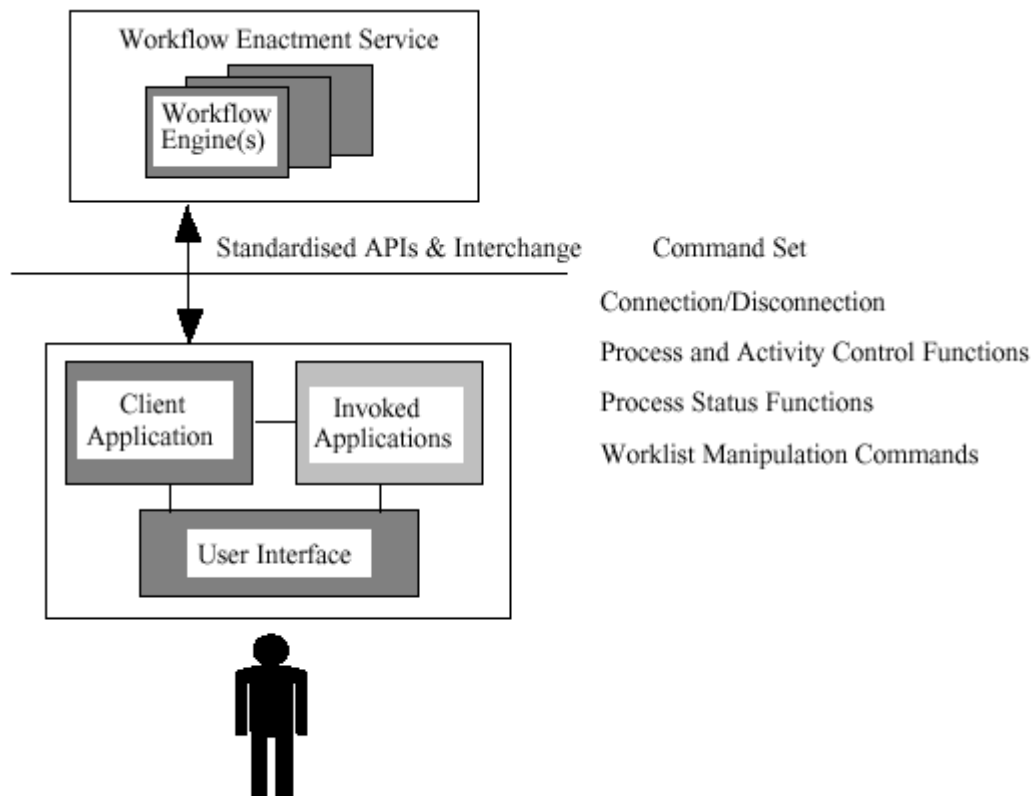


Figure 11 - Client Application Interface

Interfaz 3: Cualquier implementación particular de un WFMS no cuenta con la lógica suficiente como para invocar todas las posibles aplicaciones. El siguiente diagrama muestra el alcance de esta interfaz, la cual es aplicable a aquellas aplicaciones y application agents que fueron realizados "workflow enabled".

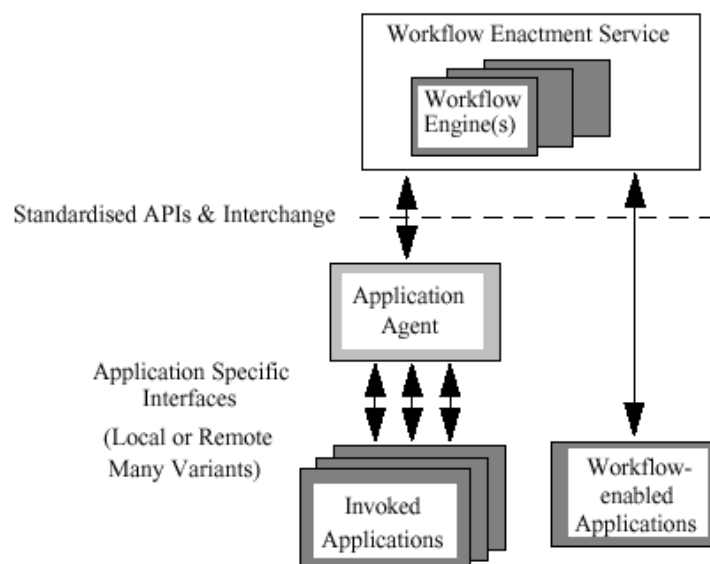


Figure 12 - Invoked Application Interface

Interfaz 4: Uno de los principales objetivos es definir estandars que permitan a los diferentes productos de workflow, pasar work ítems desde uno hacia otro. Por lo menos cuatro diferentes escenarios han sido cubiertos por la coalición, los cuales serán discutidos en futuras versiones.

El flujo de la información entre workflows heterogéneos es mostrado en la siguiente figura:

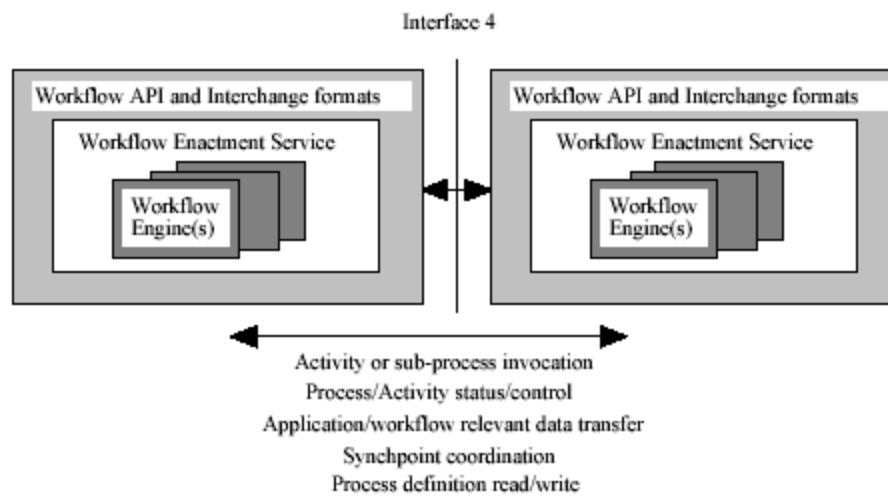


Figure 17 - Workflow interoperability interface

Interfaz 5: La interfase mostrada en la siguiente figura, muestra una aplicación independiente, interactuando con diferentes workflows, donde diferentes escenarios son también factibles. Esta interfaz incluirá funciones específicas para manipular, administrar y monitroear

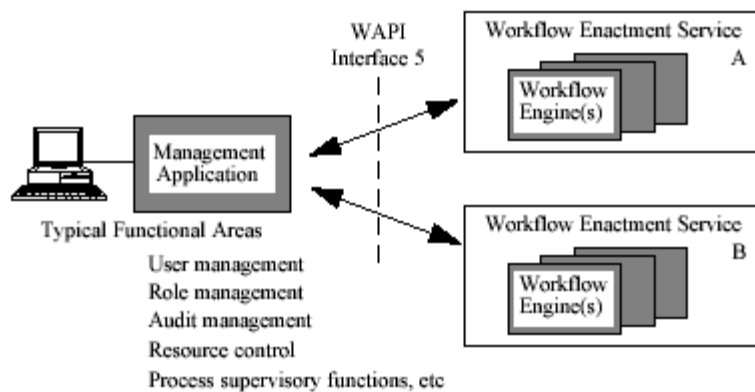


Fig 19 - Systems Administration & Monitoring Interface

2 INTERFASE 1 - INTERCAMBIO DE DEFINICIÓN DE PROCESOS.

2.1 Abstracto.

La WfMC identificó cinco interfases funcionales, como parte del programa de estandarización. Este documento refiere a la especificación de la interfase 1.

Una variedad de herramientas son usadas para analizar, modelar, describir y documentar procesos de negocio. La interfaz de definición de procesos define un formato común de intercambio, que permite transferir definiciones de procesos entre diferentes productos. Permitiendo utilizar una definición de procesos realizada por una herramienta de modelado como entrada en diferentes sistemas de Workflow.

Este formato estandarizado describe una documentación formal de un proceso. Para proveer un método común para describir definiciones, fue creado un modelo de definición. Este meta-data modelo identifica las entidades más usadas en una definición de proceso, un conjunto de atributos describen las características de este limitado conjunto de entidades. Basados en este modelo, herramientas específicas pueden transferir modelos utilizando el formato de intercambio establecido.

Este documento describe el meta-modelo que es usado para definir objetos y atributos involucrados en la definición de procesos, el XPDL.

Se requieren dos interfaces:

- 1- Una que permita importar una definición desde una cadena de caracteres a la representación interna.
- 2- Otra que permita exportar una definición desde una representación interna a una cadena de caracteres de acuerdo al lenguaje común de definición de procesos.

2.2 Introducción.

Una definición de procesos puede ser definida como:

La representación de un proceso de negocio de una forma que permita manipulación automatizada, como ser modelado y ejecución por un Sistema de Workflow. La definición de procesos consiste en una red de actividades y relaciones, criterios que indican el inicio y la terminación del proceso y información acerca cada actividad, como ser, sus participantes, aplicaciones asociada, etc

Es la definición de procesos la interpretada por el motor del Workflow. La definición de procesos puede contener referencias a subprocesos, definidos separadamente, que forman parte del proceso.

Una definición de procesos debe contener la menos un conjunto mínimo de objetos y atributos para su ejecución. Algunos de estos objetos y atributos serán heredados por cada instancia creada del proceso. APIs son para permitir operaciones sobre los atributos de las instancias y para manipular los objetos.

2.3 Intercambio de Definición de Procesos.

Esta especificación utiliza XML como el mecanismo de definición de procesos. XPDL conforma un estándar para el intercambio que permite a los productos soportar diferentes representaciones internas de definición de procesos mediante funciones para exportar e importar, que permiten mapear del estándar a la representación interna. Una variedad de mecanismos pueden ser utilizados para transferir las definiciones de procesos entre diferentes sistemas según las características de los diferentes escenarios. En cualquier caso la definición de procesos debe ser expresado de forma consistente, mediante un conjunto común de objetos, relaciones y atributos.

Los motivos para transferir definición de procesos, total o parcialmente, ocurren por diferentes motivos:

- El uso de diferentes herramientas para modelar, analizar, simular y documentar un proceso de negocio antes de una ejecución, permiten la selección del Workflow y de las herramientas de modelado, permitiendo el uso del mejor producto para cada parte del sistema.
- La transferencia y recuperación de definiciones de procesos desde / hacia un repositorio común, accedidas por diferentes herramientas o sistemas de ejecución, es deseable para proveer un único punto de control de definiciones de procesos.
- La modificación de definiciones de procesos desde un motor de Workflow hacia otro es necesario para facilitar la interoperabilidad de los procesos.

La interfase definición de procesos , define un formato común de intercambio, que permite transferir una definición de proceso entre diferentes producto para satisfacer diferentes escenarios. La Figura 1 muestra los principios de los intercambios de definiciones de procesos.

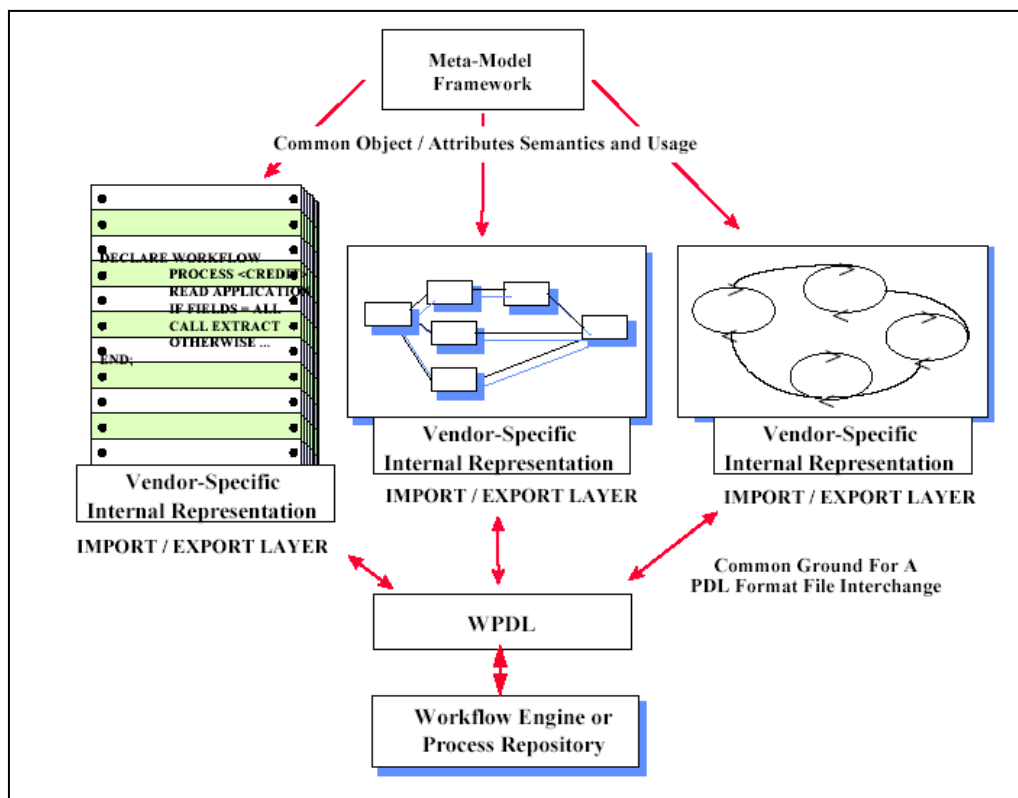


Figura 1

2.4 Especificación brindada por la WfMC.

La especificación de la WfMC para importar / exportar definiciones de procesos involucra:

- 1- El meta-modelo de definición de proceso que describe los objetos principales, sus relaciones y atributos. Este meta-modelo mínimo identifica aquellas entidades usadas en la definición de procesos y describe su semántica.
- 2- El Workflow Process Definition Language (XPDL) quien provee un lenguaje formal para la definición e intercambio de definiciones de procesos, usando objetos y atributos definidos en el meta-modelo.
- 3- Las WAPIs proveen mecanismos de manipulación de entidades y atributos, definidas en la WfMC Client Application API Specifications (WAPI) (WfMC-TC-1009).

Esta separación en meta-modelo, XPDL y APIs permiten intercambiar procesos mediante diferentes técnicas, cuando sea apropiado.

2.4.1 Meta-Modelo.

El meta-Modelo describe las principales entidades involucradas en la definición de un proceso, sus relaciones y atributos (incluyendo aquellos definidos para simulación).

Las principales entidades son mostradas en la Figura 2:

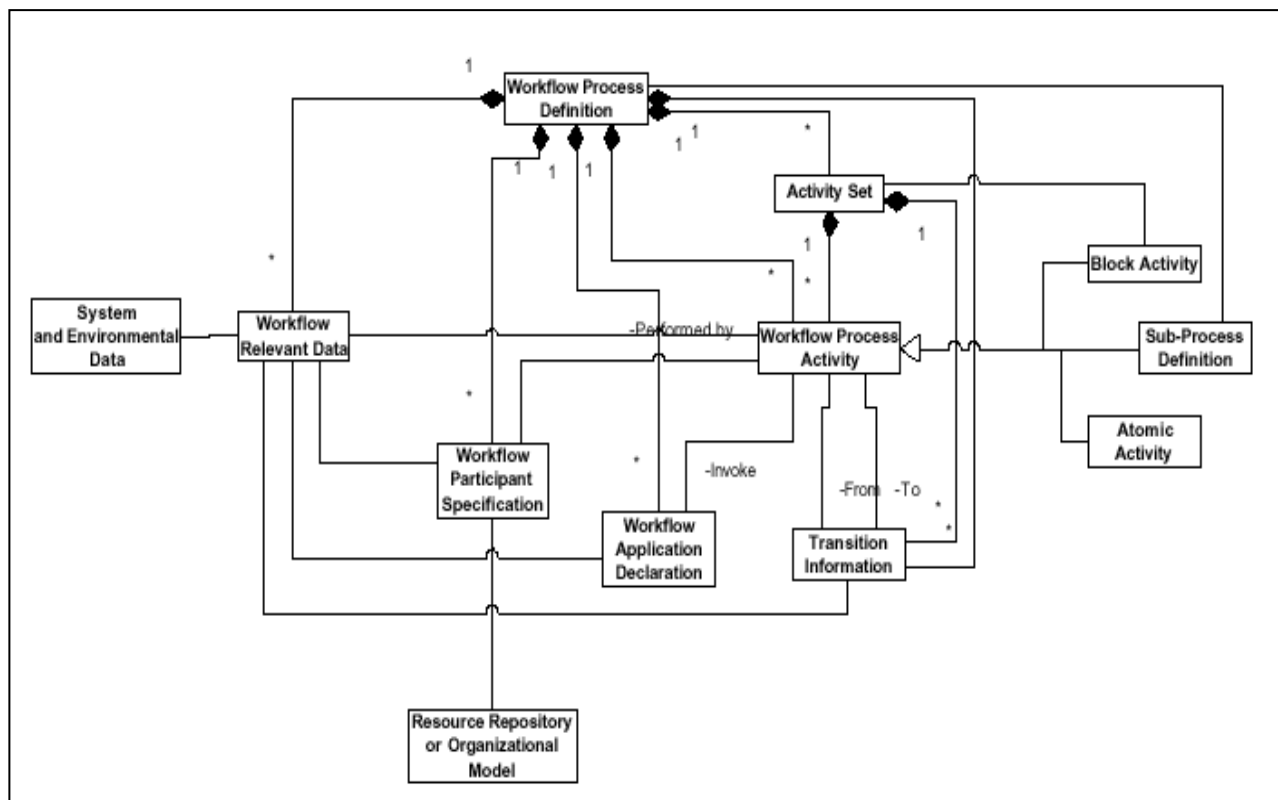


Figura 2

Para cada una de las entidades existe un conjunto de atributos asociados que describen las características de la entidad. A continuación describiremos esas entidades y atributos con más detalle. Es posible que se

requieran características que no estén consideradas en el modelo, es por esto que existen los “extended attributes”.

2.4.2 Revisión de Entidades.

El modelo identifica un conjunto de entidades básicas y sus atributos, usadas para el intercambio de definición de procesos.

- **Definición del Proceso**

Describe el proceso en sí, por ejemplo, su identificación, descripción textual, y provee otra información asociada a la administración de la definición del proceso (fecha de creación, autor, etc.) o para ser usada a nivel de proceso durante su ejecución (parámetros iniciales, prioridad, participante, atributos de simulación, etc.) Por lo tanto provee información para el proceso y para todas las entidades involucradas en el mismo.

- **Actividad del Proceso**

Una definición de procesos consiste en una o más actividades, cada una de las cuales contiene una unidad lógica de trabajo en la definición del proceso. Una actividad representa trabajo que debe ser procesado por una combinación de recursos (especificados en la asignación de participantes) o por una aplicación de computadora. También se puede agregar otra información extra como ser fecha de inicio y fin o la prioridad a ser considerada por el sistema de Workflow.

- **Información de Transición**

Las actividades están relacionadas entre si por condiciones que controlan el flujo (información sobre las transiciones). Cada transición tiene tres propiedades fundamentales, actividad inicial, actividad destino, y una condición que indica si la transición es realizada o no. Las transiciones también pueden indicar si las actividades se ejecutarán secuencialmente o en paralelo.

Transiciones más complejas, que no pueden ser expresadas utilizando transiciones elementales, pueden ser realizadas utilizando las denominadas transiciones tontas “dummy”, las cuales pueden ser utilizadas como pasos intermedios entre actividades reales

- **Declaración de Participantes**

Provee descripciones de los recursos que actúan como ejecutores de las diferentes actividades en el proceso. Los recursos particulares, que pueden ser asignados para realizar una actividad específica, son especificados como atributos de la actividad. No necesariamente se hace referencia a una persona en particular, pero debe identificar un conjunto de personas o recurso con una característica particular.

- **Repositorio de Recursos**

En escenarios más sofisticados, la declaración de participantes puede referirse a un modelo organizacional externo al sistema de Workflow. En la primera versión de la especificación no se incluye un estándar para señalar dichas referencias.

- **Declaración de Aplicaciones**

Provee una descripción de las aplicaciones que son invocadas por el Workflow para asistir o automatizar el proceso asociado a cada actividad. Identificadas en la actividad por un atributo que indica la misma. Estas aplicaciones pueden ser herramientas genéricas, procedimientos, etc.

- **Tipos Relevantes**

Indica un conjunto de tipos simples o complejos que pueden ser usados en cualquier lado de la definición del proceso, relevant data, extended attributes, etc.

- **Información Relevante**

Define la información que es creada y usada en cada instancia del proceso. Esta información debe estar disponible para las actividades y aplicaciones ejecutadas y debe ser usada para transferir información persistente o resultados intermedios entre actividades y para la evaluación de expresiones condicionales como ser en las transiciones. Las actividades, transiciones y aplicaciones invocadas, deben referir a la información relevante (Relevant Data).

- **Tipos de Datos y Expresiones**

El meta-modelo asume un conjunto estandarizado de tipos de datos: string, integer, flota, date, etc. Las expresiones son realizadas utilizando estos tipos de datos, para proveer evaluaciones condicionales.

Las entidades anteriores contienen atributos que brindan un mecanismo para la descripción de definición de procesos y constituye un Modelo Mínimo. Este puede ser expandido utilizando atributos extendidos.

2.4.3 Definición de procesos y paquetes.

Como se indica en la Figura 2, el modelo incluye varias entidades cuyo alcance puede ser mayor que una única definición de procesos. En particular las definiciones de participantes y aplicaciones pueden ser referenciados de varias definiciones de procesos.

El meta-modelo asume el uso de un repositorio común asociado con el WfMS, que contenga los diferentes tipos de entidades de la definición del proceso. En el repositorio y con la finalidad de hacer eficiente la transferencia de información desde y hacia el repositorio, se introduce el concepto de modelo de Workflow, que actúa como un contenedor que agrupa la información de entidades comunes entre diferentes definiciones de procesos, para evitar redefinir cada entidad en cada proceso individual.

Un paquete provee un contenedor que mantiene un conjunto de atributos comunes a la definición de proceso. Cada definición de proceso contenida en el paquete, heredará automáticamente los atributos comunes del modelo, a no ser que sean re-especificadas localmente en la definición del proceso.

En un paquete, el alcance de las definiciones de algunas entidades son globales y pueden ser referenciadas desde todas las definiciones de procesos contenidas en el paquete, estas entidades son:

- especificación de participantes
- declaración de aplicaciones, y
- información relevante

El paquete permite referenciar las siguientes entidades de paquetes externos:

- Identificadores de sub-flujos.
- Especificación de participantes.
- Declaración de aplicaciones.

La convención de nombres e Identificadores entre diferentes paquetes dentro del mismo repositorio para alcanzar la unicidad necesaria es conseguida buscando la primer ocurrencia dentro del modelo, de no haberla, se busca la primer ocurrencia en los modelos externos.

La estructura global de la interfase para importar y exportar definiciones de procesos y sus relaciones con el servicio de Workflow se muestra en la Figura 3.

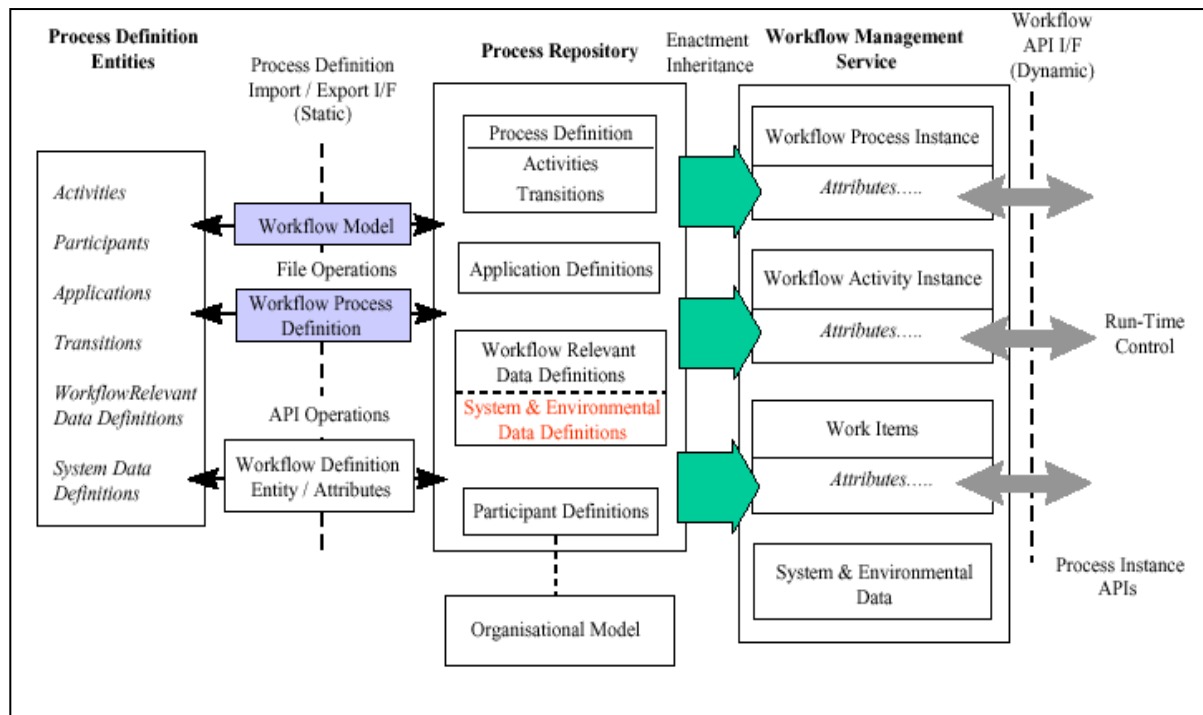


Figura 3.

2.4.4 Meta-Modelo de Procesos.

Como ya se mencionó el meta modelo define un conjunto de entidades y atributos para el intercambio de definición de procesos. Para definir un proceso, las siguientes entidades deben ser definidas, ya sea explícitamente, por herencia, o mediante una referencia cruzada a otro paquete.

- Actividades del proceso
- Información de las transiciones
- Participantes
- Declaración de aplicaciones
- Información relevante

Estas entidades contienen atributos que conforman un mecanismo para la descripción de procesos (Ver Figura 2)

2.4.5 Meta-modelo de Paquetes.

Múltiples definiciones de procesos son definidas conjuntamente en una definición de modelo. El paquete actúa como un contenedor para agrupar definiciones de procesos individuales y la información de sus entidades que es aplicable a todas las definiciones de procesos contenidas en el mismo. El meta modelo de un paquete contiene las siguientes entidades:

- Definición de procesos
- Participantes
- Declaración de aplicaciones
- Información relevante

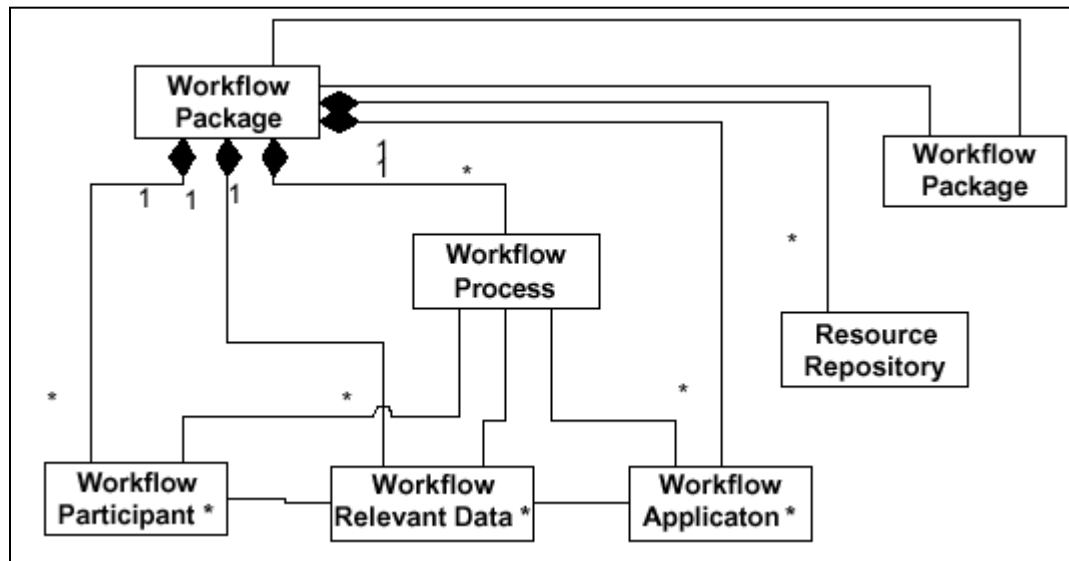


Figura 4.

Este meta modelo identifica el conjunto de entidades y atributos para el intercambio, o almacenamiento de procesos. Este define varias reglas de herencia para asociar definiciones individuales de procesos con definiciones de participantes, aplicaciones, etc. que pueden estar definidas a nivel de paquete lugar de definirse a nivel de proceso.

La definición de paquetes permite especificar atributos de las definiciones de proceso, que luego se aplicaran a todos los procesos dentro del mismo. Estos atributos no tienen necesidad de ser redefinidos en cada proceso, de serlo, este valor toma precedencia sobre el valor global.

2.4.6 Repositorio de Procesos.

La interfase para importar / exportar definiciones de procesos debe operar desde un repositorio donde se encuentran las mismas. Esta interfase se basa en la transferencia de XPDL desde y hacia dicho repositorio. Dicha interfaz permite importar o exportar información de procesos a nivel de procesos individuales y paquetes. La interfase entre el repositorio y el sistema de Workflow es especifica de cada fabricante y no forma parte de este estándar. Tampoco forma parte del estándar la estructura interna del repositorio.

2.4.7 Alcance y redefinición.

La posibilidad de redefinir atributos, entidades y referenciar paquetes externos introducen los principios de alcance y jerarquía.

- 1- Relevant Data: (Información relevante) tiene un alcance definido por la entidad y no se anida.
- 2- Atributos: Los atributos, incluyendo los extendidos, pero puede anidarse, es decir, pueden ser referenciados en niveles inferiores.
- 3- Participantes y aplicaciones: tiene un visibilidad y alcance equivalente a los atributos. La información relevante y los atributos extendidos deben ser definidos donde serán usados, al menos en el mismo paquete..

2.4.8 Revisión de entidades.

Package	Workflow Process	Activity	Transition	Application	Data Field (Workflow Relevant Data)	Participant
- Id - Name - Description - Extended Attributes - XPDL Version - Source Vendor ID - Creation Date - Version - Author - Codepage - Country Key - Publication Status - Conformance Class - Priority Unit	- Id - Name - Description - Extended Attributes - Creation Date - Version - Author - Codepage - Country Key - Publication Status - Priority - Limit - Valid From Date - Valid To Date	- Id - Name - Description - Extended Attributes - Automation Mode - Split - Join - Priority - Limit - Start Mode - Finish Mode - Deadline	- Id - Name - Description - Extended Attributes	- Id - Name - Description - Extended Attributes	- Id - Name - Description - Extended Attributes - Data Type	- Id - Name - Description - Extended Attributes - Participant Type
- Responsible - External Package	- Parameters - Responsible	- Performer - Tool - Subflow - ActivitySet - Actual Parameter	- Condition - From - To	- Parameters	- Initial Value	
- Documentation - Icon	- Documentation - Icon	- Documentation- -Icon				
- Cost Unit	-Duration Unit - Duration - Waiting Time - Working Time	-Cost - Duration - Waiting Time - Working Time				

2.4.9 Extensiones específicas.

Si bien el meta-modelo y el XPDL asociado contiene la mayoría de los constructores requeridos para el intercambio de definiciones de procesos, pueden haber circunstancias en las cuales información adicional sea necesaria en la definición del proceso.

El mecanismo que abajo descrito provee un forma estandarizada para expresar extensiones con el propósito del intercambio, pero requiere un adaptación del sistema para que este pueda comprender la información en tiempo de ejecución.

Extended Attributes.

El método para expresar extensiones se realiza mediante el uso de atributos extendidos. Estos atributos son definidos por el usuario o fabricante, para expresar cualquier información adicional que debe ser intercambiada entre sistemas. Cualquier semántica asociada con el uso de atributos extendidos durante la ejecución del proceso, es especificada separadamente, y requiere un acuerdo bilateral entre el exportador y el importador.

Lenguaje de definición de procesos en XML. (XPDL)

Elementos comunes para múltiples entidades.

Enumeraremos aquellos elementos comunes a las diferentes entidades, señalando sus principales características...

- **Extended Attributes:** Pueden ser utilizados en todas las entidades. Permite a los fabricantes extender las funcionalidades para satisfacer necesidades personales.
- **Formal Parameters:** Pueden ser utilizados como atributos en los procesos y aplicaciones. Son pasados durante la invocación.

El pasaje de parámetros se define de la siguiente forma.

- a- Cualquier parámetro de solo lectura (IN), es inicializado con el valor del actual parameter correspondiente
- b- Cualquier parámetro de lectura / escritura (INOUT), es inicializado con el valor del actual parameter correspondiente, que debe ser el identificador de una entidad de la relevant data.
- c- Cualquier parámetro de solo escritura (OUT) es inicializado en zero.

La copia y recuperación de parámetros es tratado como operaciones atómicas, para evitar conflictos de acceso de operaciones concurrentes sobre la relevant data.

El mapeo de actual parameters a formal parameters durante la invocación es definido por una lista de mapeo de parámetros. Los actual parameters son mapeados 1:1. A los parámetros formales, en secuencia. Es requerida la compatibilidad de tipos entre las definiciones y debe ser forzada en tiempo de ejecución por el sistema de Workflow.

- **External Reference:** Es una referencia a una entidad externa. Puede ser utilizada en data types, participantes, aplicaciones.

Definición de Paquetes.

Es posible definir varios procesos en un paquete, que pueden compartir las mismas herramientas y participantes. Es aconsejable construir un paquete por proceso de negocio que contenga todos los procesos necesarios, así como las herramientas y participantes aunque no sean requeridos.

Los elementos que conforman la definición de un paquete son los siguientes:

- **Package definition header:** Contiene toda la información central del paquete, como ser: versión, fabricante, etc.
- **Redefinable header:** Cubre aquellos atributos que deben ser redefinidos en el cabezal de la definición de algún proceso del paquete.
- **Conformance class:** Describe la estructura de la red, es decir, si puede tener loops, si pueden estar anidados, etc.
- **Script:** Identifica el lenguaje de script utilizado en las expresiones XPDL. Una expresión puede ser utilizada siempre que el elemento sea de tipo xsd:string Se puede, por ejemplo, usar una expresión en los actual parameters.

Declaración de Aplicaciones

Es una lista de todas las aplicaciones o herramientas requeridas e invocadas por los procesos. La verdadera definición de las herramientas o aplicaciones no es necesaria, alcanza con mencionarlas (nombrarlas) y deben ser manejadas por un administrador de objetos. La razón es para permitir el manejo de diferentes plataformas, donde diferentes programas deben ser invocados para cada una. XPDL abstrae la implementación o ambiente concreto, ya que no es necesario en tiempo de definición.

- **Parámetros de invocación:** La declaración de aplicaciones debe contener definición de parámetros para la invocación, y debe poder utilizarlos en otras invocaciones

Definición de Procesos.

Contiene definiciones o declaraciones para actividades, transiciones, aplicaciones y relevant data. Los atributos deben ser especificados para la administración, como ser: autor, versión; para tiempo de ejecución: prioridad.

Enumeraremos los principales componentes del Workflow process definition:

- **Workflow process definition header:** Mantiene la información específica de la definición como ser: versión, duración, prioridad, etc.
- **Workflow process redefinable header:** Permite mantener la información que puede ser redefinida.
- **Activity Set:** Representa un conjunto de actividades y transiciones, las mismas deben referir solo a actividades en el mismo conjunto, estas actividades pueden ser solo ejecutadas por activities block.

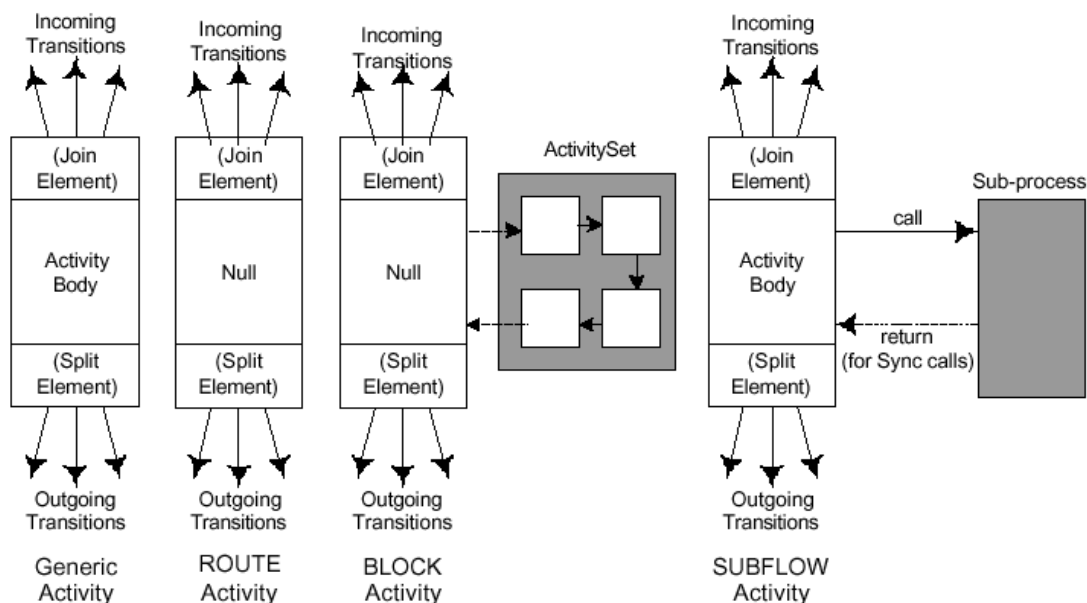
Actividades de Proceso.

Esta entidad es usada para definir aquellas actividades que debe ejecutar el sistema. Los atributos deben ser definidos para especificar el control, diferentes alternativas, asignación de participantes, información de tiempo de ejecución como ser prioridad, etc. Otros atributos pueden depender del tipo de actividad.

La siguiente tabla describe el uso de los atributos en los diferentes tipos de actividades:

Entity Types (usage within Activity Type)	Activity Type				BlockActivity
	Implementation Type			Route	
	None	Application	Subflow		
Transition Restriction	Normal	Normal	Normal, plus subflow call / return within activity	Normal; any additional controls implemented within Route activity	Normal; refers to activities within same context, not to activities within ActivitySet
Participant Assignment	Normal	Normal	N/A	N/A	N/A
Application Assignment	None	Yes	N/A	N/A	N/A
Use of workflow Relevant Data	Normal	Normal	May be used in parameter passing	May be used in routing control conditions	May be used in routing control conditions

En el siguiente diagrama se muestra la estructura genérica de una actividad y sus variantes:



Cuando el tipo de implementación es NONE, la actividad es controlada manualmente y su completitud debe ser explícitamente señalada al sistema de Workflow. Estas actividades comprenden generalmente instrucciones a participantes quienes deben realizar tareas no automatizadas.

Cuando la actividad es de tipo sub-flujo, cualquier parámetro de entrada pasado al sub-proceso llamado, debe ser declarado como relevant data en el proceso llamado.

Actividades de tipo Routing o Block no manipulan Workflow relevant data directamente, pero pueden referir la misma con expresiones condicionales.

Actividades "Route".

Este tipo de actividad es denominada actividad "tonta" que permite generar expresiones en cascada en las condiciones de transición.

Actividades "Block".

Una actividad de este tipo ejecuta un ActivitySet. Desde una actividad Block, la ejecución procede desde la primera actividad en el conjunto, hasta alcanzar la actividad de salida (actividad sin transiciones salientes). Luego la ejecución continúa con las transiciones salientes del Activity Block.

- **Atributos de Control de ejecución.**

Estos atributos, permiten la definición de propiedades específicas de la actividad de la ejecución.

- Modo automático: es completamente controlado por el motor del Workflow.
- Modo manual: requiere la interacción con el usuario para que la actividad comience y finalice.

- **Alternativas de implementación.**

Se asume que la ejecución de una actividad es atómica respecto a la información manejada por el motor del Workflow. Esto implica que en caso que el sistema aborta o cancele la ejecución de la actividad, la relevant data, y la información de control debe ser recuperada.

Una actividad puede ser implementada en cualquiera de las maneras descritas a continuación.

- **No Implementation.**

Significa que la implementación de esta actividad no es soportada automáticamente por el Workflow utilizando aplicaciones o procedimientos. Dos alternativas fueron identificadas.

- Es una actividad manual
- Es una actividad implícita, lo que implica que sea conocida por el motor del Workflow, como ser los extended attributes del fabricante.

- **Tool.**

Esta actividad es implementada por una o más herramientas. Una herramienta puede ser un programa, invocado vía interfaz 2 y 3.

- **Subflow.**

Un sub-flujo puede ser ejecutado sincrónica o asincrónicamente. En el caso de la ejecución asincrónica, no se permite el retorno de parámetros de los procesos llamados. La sincronización con el sub-flujo iniciado, si es requerido, debe hacerse de otras formas como ser eventos, no destritos en este documento. En caso de ejecución sincrónica, la ejecución de la actividad es suspendida hasta la finalización del mismo, donde la actividad es resumida.

Relación con participantes.

La relación entre una actividad y un potencial realizador de la misma, esta dado por el atributo que indica el participante asignado. Este provee un link al participante.

Cuando una expresión evalúa en un conjunto vacío de participantes o está formado por más de uno, la decisión debe ser tomado por el motor del Workflow en tiempo de ejecución, dicha decisión está fuera del alcance de la especificación.

Información de Transiciones.

Describe la información de las posibles transiciones entre actividades y condiciones que habilitan o deshabilitan las mismas durante la ejecución. La definición de un proceso puede ser vista como una red donde las actividades son los nodos.

La condición habilita o no una determinada transición, de no haber, evaluará en verdadero. Si existen múltiples transiciones entrantes o salientes, más restricciones deben ser evaluadas, (AND/OR en sus variantes SPLIT/JOIN). Es posible sincronizar múltiples hilos mediante el uso de split y join.

Participantes.

Un participante puede ser uno de los siguientes tipos: recurso, conjunto de recursos, unidades organizacionales, humanos, sistemas. La definición es una abstracción entre el verdadero participante y la actividad a ser realizado. Durante la ejecución estas abstracciones deben ser evaluadas y asignadas el recurso concreto.

Un repositorio externo o modelo organizacional puede ser referenciado, donde se puede obtener información adicional que complemente los participantes básicos presentados.

Información Relevante

Esta información representa la variables de la definición de procesos. Esta información es utilizada en general para mantener información de decisión, utilizada en condiciones que es utilizada entre actividades. Esta información debe ser diferenciada de la información de aplicaciones, la que es manejada o accedida por la propia aplicación y no es accesible para el sistema de Workflow. El tipo de dato es identificado por el atributo DataType, y le dice al sistema la información necesaria para definir el objeto apropiado para almacenar la información, que es manejada por la instancia del proceso.

Tipos de Datos

Consiste en un conjunto de tipos estándar que pueden ser usados como parte de la especificación de la información del proceso, parámetros, etc.

Tipos básicos:

- String
- Float
- Integer
- Reference
- DateTime
- Boolean
- Performer

Tipos Complejos: XPDL permite definir tipos complejos como ser registros, uniones, enumerados, etc. Dichos tipos son definidos usando SchemaType, donde se define el esquema del mismo.

Es posible rehusar tipos complejos, definiendo el tipo utilizando TypeDeclaration y luego referirlo mediante el tipo DeclaredType.

3 INTERFASES 2 Y 3 – INTERFASE DE APLICACIONES DE WORKFLOW

3.1 Propósito

El propósito de este documento es especificar un estándar de interfases de administración de workflows que son llamadas API ("Application Programming Interfaces").

Las invocaciones a la API proveen un método consistente de acceso a funciones de workflow.

El conjunto de todas las API es llamado WAPI ("Workflow Application Programming Interfaces").

Este documento define la especificación de la API de la "Workflow Management Coalition" para la construcción de aplicaciones de workflow.

3.2 Perspectiva General

Las implementaciones de estas invocaciones a las API son también un intento para permitir que las aplicaciones de workflow se puedan ajustar para operar con diferentes motores de workflow usando esta interfase de API en común.

Las invocaciones a las WAPI son hechas en tiempo de ejecución. Esto es cuando procesos se están ejecutando o están por ser ejecutados.

A través de este conjunto de funciones, la WAPI provee el conjunto de servicios de workflow que un "Workflow Enactment Service" provee.

Las funciones de un motor de workflow pueden ser clasificadas en las siguientes áreas:

- Funciones de conexión
- Funciones de definición de workflows
- Funciones de control de procesos
- Funciones de control de actividad
- Funciones de estado de proceso
- Funciones de estado de actividad
- Funciones de lista de trabajo ("worklist")
- Funciones de administración

3.3 Definición de Interfases de Aplicaciones

La interfase de la "Workflow Management Coalition" para invocar aplicaciones no define directamente un mecanismo de control de aplicaciones.

Hoy en día, los consumidores y fabricantes de workflows se confrontan con distintos sistemas operativos y mecanismos de comunicación de aplicaciones. Por esto los sistemas de workflow necesitan una interfase para especificar drivers para las aplicaciones. Con la definición de estos drivers para invocar y controlar las aplicaciones, la "Workflow Management Coalition" ofrece una interfase que permite un protocolo estandarizado entre productos de workflow y cualquier otro sistema.

3.4 Propósito

La interfase de invocación de aplicaciones define un mecanismo de interfase entre un sistema de administración de workflow y cualquier otra aplicación.

Invocar una aplicación no es una funcionalidad específica de un workflow, pero un sistema de workflow no va a tener mucho sentido sin esta funcionalidad.

Basados en diferentes tecnologías de comunicación los llamados "Tool Agents" pueden manejar el control de una aplicación y el intercambio de información. Estos "Tool Agents" representan por lo menos una tecnología específica de invocación.

Por ejemplo un "Tool Agent" soporta comando DDE, otro se comunican basados en protocolos como OLE o CORBA.

La tecnología para interactuar entre un "Tool Agent" y una aplicación depende de la arquitectura y de la interfase específica de la aplicación, la cual debe ser manejada bajo el control del mismo "Tool Agent". La interfase sugerida define la manera en que el "Tool Agent" puede ser usado por una aplicación de workflow, por ejemplo un manejador de listas de trabajo ("worklist handler") o el motor de workflow. Finalmente, el propósito de los "Tool Agents" puede ser comparado con el propósito de componentes de software estandarizados.

3.5 Descripción de las WAPI

Las WAPI están agrupadas de la siguiente forma:

- Funciones WAPI de Conexión
- Funciones WAPI de Control de Procesos
- Funciones WAPI de Control de Actividades
- Funciones WAPI de Estado de Procesos
- Funciones WAPI de Estado de Actividades
- Funciones WAPI de listas de trabajo ("worklists")
- Funciones WAPI de Administración
- Funciones WAPI de invocación de aplicaciones

3.6 Funciones WAPI de Conexión

Hay 2 funciones que se usan para conexión / desconexión, estas son **WMConnect** / **WMDisconnect**. Sirven para limitar un conjunto de trabajo relacionado por la aplicación que lo usa.

WMConnect retorna un "handle" cuyo valor es usado en toda otra invocación a API.

Información dada por el WMConnect incluye información de identificación relacionada a que/quien esta requiriendo servicios del motor para uso de un servicio autenticado.

3.7 Operaciones entre la API y el motor

En la construcción de las invocaciones a las API, se intenta tener poco impacto sobre la estructura operacional de como un determinado producto de workflow las implementa.

Diferentes tipos de mecanismos pueden ser empleados para entregar el pedido al motor.

Un método particular de interacción entre un motor y las llamadas a la API puede ser por ejemplo RPC, también se puede usar "messaging" u otros métodos.

Si se usa el mecanismo de "messaging" y el motor es remoto, el WMConnect puede resultar en el establecimiento de que cola de mensajes se usa para la interacción entre las API y el motor.

Si una base de datos se usa, uno de los resultados del WMConnect puede ser el establecimiento de una conexión a la apropiada facilidad de almacenamiento de datos.

Un workflow en particular puede elegir entre aceptar el WMConnect, retornar un "handle" e ignorar el hecho de que ocurrió.

En algunos casos un producto puede requerir conectar un solo workstation a varios motores. Es posible que varios WMConnect estén activos concurrentemente y los subsecuentes comandos API sean direccionados al motor correcto. El WMConnect puede ser usado para designar un motor particular. El "handle" retornado del connect puede ser usado en subsecuentes invocaciones a API para ligar a estos al motor que los va a ejecutar.

El resultado de un WMDisconnect puede ser muy variado dependiendo de la implementación del producto.

Su propósito es el de indicar que la aplicación que estaba haciendo las invocaciones a las API no va a acceder mas a las funciones del motor con el contexto previo.

3.8 Como opera una aplicación cuando usa invocaciones a la API

La estructura operacional de una aplicación, al estar invocando la API, es afectada por la forma en que dicha API están construidas. La construcción actual de las invocaciones a la API resulta en que el segmento de la aplicación hace la invocación a la API para que corra en modo bloqueado ("blocked mode"). Esto es, la aplicación hace la invocación a la API y se queda esperando por una respuesta.

Cuando se hace la invocación a la API, la aplicación le pasa el control de la ejecución a la API y no retoma dicho control hasta que la API termina.

Las API pudieron haberse hecho de tal forma que la aplicación haga la invocación en modo desbloqueado ("unblocked mode"). Esto es, hacer que la invocación a la API retorne enseguida en vez de quedarse esperando a que termine.

De haberse hecho las API de esta forma la Coalición necesitaría definir funciones adicionales para soportar modos de operación "sin conexión". Los comandos API WMConnect /WMDisconnect como están ahora definidos no tienen nada que ver con la habilidad de una aplicación de correr conectada o desconectada.

3.9 Funciones WAPI de control de procesos

Pueden ser definidas como aquellas que cambian el estado operacional de una o más instancias de procesos.

<i>METODO</i>	<i>DESCRIPCION</i>
WMOpenProcessDefinitionsList	Devuelve una lista de todas las definiciones de procesos que cumplen con el criterio de un filtro que se pasa por parámetro. Este método puede ser usado por administradores de procesos para ver todas las definiciones de procesos y así ver que procesos pueden ser empezados por determinadas personas.
WMFetchProcessDefinition	Devuelve la próxima definición de proceso del conjunto de definiciones de procesos que devolvió el método WMOpenProcessDefinitionsList
WMCloseProcessDefintionsList	Cierra la consulta sobre las definiciones de procesos
WMOpenProcessDefinitionStatesList	Abre la consulta que produce la lista de estados de definiciones de procesos que cumplen el criterio de un filtro. Uno de los usos de esta API es la de, conjuntamente con las llamadas a fetch y close, permitir a una aplicación de workflow consultar al motor sobre los estados disponibles de las definiciones de procesos que cumplen con el filtro para así ofrecer esta lista al usuario de la aplicación.
WMFetchProcessDefintionState	Devuelve el próximo estado de definición de proceso, de la lista de estados de definiciones de procesos que machean con el criterio del filtro.
WMCloseProcessDefinitionStatesList	Cierra la consulta para los estados de las definiciones de procesos.
WMChangeProcessDefinitionState	Cambia el estado de una definición de proceso
WMCreateProcessInstance	Crea una instancia de una definición de proceso previamente definido. Una instancia operacional de la definición del proceso nombrado va a ser creado por el motor como resultado de esta API. Un llamado a WMStartProcess va a empezar la ejecución del proceso.

WMStartProcess	Arranca el proceso nombrado. El motor empieza a ejecutar el proceso del cual una instancia se creo. Cuando un proceso es empezado a través de este comando, la primera actividad del proceso se va a arrancar. El id de la instancia de proceso devuelta por esta llamada va a ser valida durante el tiempo de vida de la instancia del proceso. El programador necesita mantener la asociación entre el nuevo id de la instancia del proceso y la sesión.
WMTerminateProcessInstance	Termina la instancia de un proceso. Esta API provee la capacidad de terminar un proceso sin abortar la instancia del proceso.
WMOpenProcessInstanceStatesList	Abre una consulta que produce la lista de estados de las instancias de procesos que matchean el criterio del filtro.
WMFetchProcessInstanceState	Devuelve el próximo estado de instancia de proceso de la lista de estados de la instancia de proceso que matchean el criterio del filtro.
WMCloseProcessInstanceStatesList	Cierra la consulta para estados de instancias de procesos.
WMChangeProcessInstanceState	Cambia el estado de la instancia del proceso nombrado. La ejecución de esta API va a causar que la instancia del proceso nombrado cambie a un nuevo estado. En este caso el significado de todos los estados dependiente de la implementación del motor.
WMOpenProcessInstanceAttributesList	Abre la consulta que produce la lista de atributos que matchean el criterio del filtro.
WMFetchProcessInstanceAttribute	Devuelve el próximo atributo de la instancia del proceso de la lista de atributos que matchean con el criterio del filtro.
WMCloseProcessInstanceAttributesList	Cierra la consulta para los atributos de las instancias de procesos.
WMGetProcessInstanceAttributeValue	Devuelve el valor, tipo y largo de una instancia de proceso especificado por los parámetros
WMAssignProcessInstanceAttribute	Asigna el atributo a la instancia de proceso.

3.10 Funciones WAPI de Control de Actividades

Pueden ser definidas como aquellas que cambian el estado operacional de una o más instancias de actividades. Estas invocaciones a la API están orientadas hacia el usuario final de workflow. Sin embargo algunas de las invocaciones a la API pueden afectar múltiples usuarios y normalmente deberían estar restringidas al uso de un administrador.

METODO	DESCRIPCION
WMOpenActivityInstanceStatesList	Abre la consulta que produce la lista de estados de las instancias de las actividades que matchean con el criterio del filtro.
WMFetchActivityInstanceState	Devuelve el próximo estado de la instancia de la actividad de la lista de estados de las instancias de actividades que matchean con el criterio del filtro.
WMCloseActivityInstanceStatesList	Cierra la consulta de los estados de las instancias de actividades.
WMChangeActivityInstanceState	Cambia el estado de la instancia de la actividad nombrada. Esta API hace que el motor cambie el estado de una sola instancia de actividad en una instancia de proceso. Esto permite que el estado de la instancia de actividad se cambie sin impactar otras en la instancia de proceso.
WMOpenActivityInstanceAttributesList	Abre la consulta que produce la lista de atributos de actividades que matchean con el criterio del filtro.
WMFetchActivityInstanceAttribute	Devuelve el próximo atributo de la instancia de actividad de la lista de atributos de actividades que matchean con el criterio del filtro.
WMCloseActivityInstanceAttributesList	Cierra la consulta de los atributos de las instancias de actividades.
WMGetActivityInstanceAttributeValue	Devuelve el valor, tipo y largo de un atributo de una instancia de actividad especificado por los parámetros.
WMAssignActivityInstanceAttribute	Asigna un atributo a una instancia de actividad

3.11 Funciones WAPI de Estado de Procesos

Las funciones de estado de proceso son para proveer una vista del trabajo hecho, el trabajo a hacer, trabajo asociado con un participante de workflow, o trabajo asociado a un grupo de participantes de workflows, etc. Las consultas de estados pueden ser hechas por un participante normal de workflow o pueden ser requeridas por un administrador de procesos quien desea ver el progreso de trabajo con su dominio.

Las invocaciones a la API de estado están estructuradas de forma que proveen rangos de vistas desde una vista global de trabajo hasta una vista de trabajo con una sola instancia de proceso.

Estas vistas son de la siguiente forma:

1	Todas las instancias de procesos asociadas con una definición de proceso.	WM(Open+Fetch+Close)ProcessInstancesList
2	Una vista de una sola instancia de proceso.	WMGetProcessInstance

METODO	DESCRIPCION
WMOpenProcessInstancesList	Abre la consulta que produce una lista de instancias de procesos que matchean el criterio del filtro.
WMFetchProcessInstance	Devuelve la próxima instancia de proceso de la lista de instancias de procesos que cumplen el criterio del filtro.
WMCloseProcessInstancesList	Cierra la consulta de instancias de procesos.
WMGetProcessInstance	Devuelve un registro específico de una instancia de proceso.

3.12 Funciones WAPI de Estado de Actividades

Las funciones de estado de actividades son para proveer una vista del trabajo hecho, el trabajo a hacer, trabajo asociado con un participante de workflow, o trabajo asociado a un grupo de participantes de workflows, etc.

Las consultas de estados pueden ser hechas por un participante normal de workflow o pueden ser requeridas por un administrador de procesos quien desea ver el progreso de trabajo con su dominio.

Las invocaciones a la API de estado están estructuradas de forma que proveen rangos de vistas desde una vista global de trabajo hasta una vista de trabajo con una sola instancia de actividad.

Estas vistas son de la siguiente forma:

1	Todas las instancias de actividades asociadas con una definición de proceso o instancia de proceso.	WM(Open+Fetch+Close)ActivityInstancesList
2	Una vista de una sola actividad con una instancia de proceso.	WMGetActivityInstance

METODO	DESCRIPCION
WMOpenActivityInstancesList	Abre la consulta que produce la lista de las instancias de actividades que matchean el criterio del filtro.
WMFetchActivityInstance	Devuelve la próxima instancia de actividad de la lista de instancias de actividades que cumplen con el criterio del filtro de la correspondiente invocación a WMOpenActivityInstancesList.
WMCloseActivityInstancesList	Cierra la consulta de instancias de actividades.
WMGetActivityInstance	Devuelve el registro de una instancia de actividad específica.

3.13 Funciones WAPI de lista de trabajo ("worklist")

Proveen a los participantes de workflow acceso a información sobre trabajo al cual fueron asignados. Como se describe el modelo de referencia de la Coalición, un proceso consiste en un conjunto de actividades conectadas de forma de controlar las invocaciones a las aplicaciones. Una actividad esta asociada con una o más aplicaciones a invocar, durante tiempo de ejecución, esta asociada con la persona(s) que ha sido asignada a hacer el trabajo.

Dependiendo de la implementación, a un participante de workflow le puede ser asignado uno o más piezas de trabajos en cualquier momento. Cada pieza de trabajo asignado a un participante de workflow es llamado ítem de trabajo ("work item") y la colección de todos los ítems de trabajo ("workitems") asignados a un participante de workflow es llamada lista de trabajo ("worklist") del participante del workflow.

Nota: Para clarificar la diferencia entre actividad y "work item" se incluye la siguiente discusión. Cuando un proceso esta siendo definido, una actividad es la construcción usada para definir una pieza de trabajo a ser hecho. Durante tiempo de ejecución, cuando la actividad es lista para ser ejecutada y uno o más participantes están asignados a hacer el trabajo, se crea un ítem de trabajo ("work item") y se coloca en la lista de trabajo ("worklist") de dicho participante.

Por eso aunque una actividad y un ítem de trabajo ("work item") representan una pieza de trabajo, aparecen en diferentes momentos, puede pasar que haya mas de un ítem de trabajo ("work item") para una actividad y algunas características operacionales pueden ser diferentes. Entonces una lista de trabajo ("worklist") es una lista de ítems de trabajo ("work items") asociados a un participante de workflow. Las invocaciones a la API en esta sección existen para la manipulación de ítems de trabajo ("work items") y de listas de trabajo ("worklists").

METODO	DESCRIPCION
WMOpenWorkList	Abre la consulta que produce la lista de trabajo ("worklist") que matchea con el criterio del filtro. Esta función provee la capacidad de retornar una lista de ítems de trabajo ("workitems") asignados a un participante o grupo de workflow.
WMFetchWorkItem	Devuelve el próximo ítem de trabajo ("workitem") de la lista de ítems de trabajo ("workitems") que cumple el criterio de selección en la llamada a WMOpenWorkList
WMCloseWorkList	Cierra la consulta sobre ítems de trabajo ("workitems").
WMGetWorkItem	Devuelve el registro de un específico ítem de trabajo ("workitem").
WMCompleteWorkItem	Le dice al motor de workflow que este ítem de trabajo ("workitem") ha sido completado.
WMOpenWorkitemStatesList	Abre una consulta que produce la lista de estados de ítems de trabajo ("workitems") que matchean con el criterio del filtro.
WMFetchWorkitemState	Devuelve el próximo estado del ítem de trabajo ("workitem") de la lista de estados de ítems de trabajo ("workitems") que matchean con el criterio del filtro.
WMCloseWorkitemStatesList	Cierra la consulta de estados de ítems de trabajo ("workitems").
WMChangeWorkitemState	Cambiar el estado del nombrado ítem de trabajo ("workitem").
WMReassignWorkItem	Permite que un ítem de trabajo ("workitem") de una lista de trabajo ("worklist") de un participante de workflow pueda ser reasignado a otro participante de workflow.
WMOpenWorkItemAttributesList	Abre la consulta para producir la lista de atributos de ítems de trabajo ("workitems") que cumplen con el criterio de un filtro.

WMFetchWorkItemAttribute	Devuelve el próximo atributo de ítem de trabajo ("workitem") de la lista de atributos de ítems de trabajo ("workitems") que cumplen el criterio del filtro.
WMCloseWorkItemAttributesList	Cierra la consulta de atributos de ítems de trabajo ("workitems").
WMGetWorkItemAttributeValue	Devuelve el valor, tipo y largo de un atributo de ítem de trabajo ("workitem") especificado por uno de los parámetros
WMAssignWorkItemAttribute	Asigna un atributo a un ítem de trabajo ("workitem")

3.14 Funciones WAPI de Administración

El conjunto de funciones de administración proveen las funcionalidades necesarias para realizar funciones de administración y mantenimiento de un sistema de workflow.

Este conjunto incluye los mínimos servicios contemplados para esta interfase de aplicación cliente. El conjunto incluye funciones para cambiar el estado de un conjunto de instancias de procesos o instancias de actividades, funciones para terminar y abortar instancias de procesos y para asignar atributos a un conjunto de instancias de procesos y actividades.

METODO	DESCRIPCION
WMChangeProcessInstancesState	Cambia el estado de las instancias de la definición de proceso nombrado que matchea con el criterio del filtro.
WMChangeActivityInstancesState	Cambia el estado de las instancias de actividades de un nombre particular asociado a una definición de proceso que matchea con el criterio del filtro.
WMTerminateProcessInstances	Termina las instancias de los procesos de las nombradas definiciones de procesos que matchean con el criterio del filtro.
WMAssignProcessInstancesAttribute	Asigna el atributo correspondiente a un conjunto de instancias de procesos con una definición de proceso que matchea con el criterio del filtro.
WMAssignActivityInstancesAttribute	Asigna el atributo correspondiente a un conjunto de instancias de actividades con una definición de proceso que matchean con el criterio del filtro.
WMAbortProcessInstances	Aborta el conjunto de instancias de procesos que corresponden a la definición de proceso nombrado que matchea con el criterio del filtro.
WMAbortProcessInstance	Aborta la instancia de proceso especificada.

3.15 Funciones WAPI de invocación de aplicaciones

El conjunto de funciones de la interfase de aplicaciones proveen servicios a Tool-Agents, para invocar y controlar, aplicaciones asociadas con "work items" específicos.

La interfase de invocación de aplicaciones define un conjunto de API el cual es muy recomendado para ser usado por los componentes de un sistema de workflow (motor y aplicaciones clientes) para controlar drivers especializados de aplicaciones llamados Tool-Agents.

Estos Tool-Agents arrancan y paran las aplicaciones, pasan información hacia y desde la aplicación.

Por esto, la interfase de invocación de aplicaciones WAPI's son dirigidas contra los Tool-Agents. No obstante información adicional del workflow puede ser pedida por la aplicación vía Tool-Agents usando las funciones standard WAPI's.

Como la interfase de invocación de aplicaciones puede ser manejar pedidos bi-direccionales (pedidos desde y hacia la aplicación), depende de la interfase y de la arquitectura de las aplicaciones como interactuar con un Tool-Agent.

Esta interfase va a permitir el pedido y la actualización de datos de la aplicación, y más funcionalidad de tiempo de ejecución.

El sistema de workflow debe saber cuales son los Tool-Agents instalados.

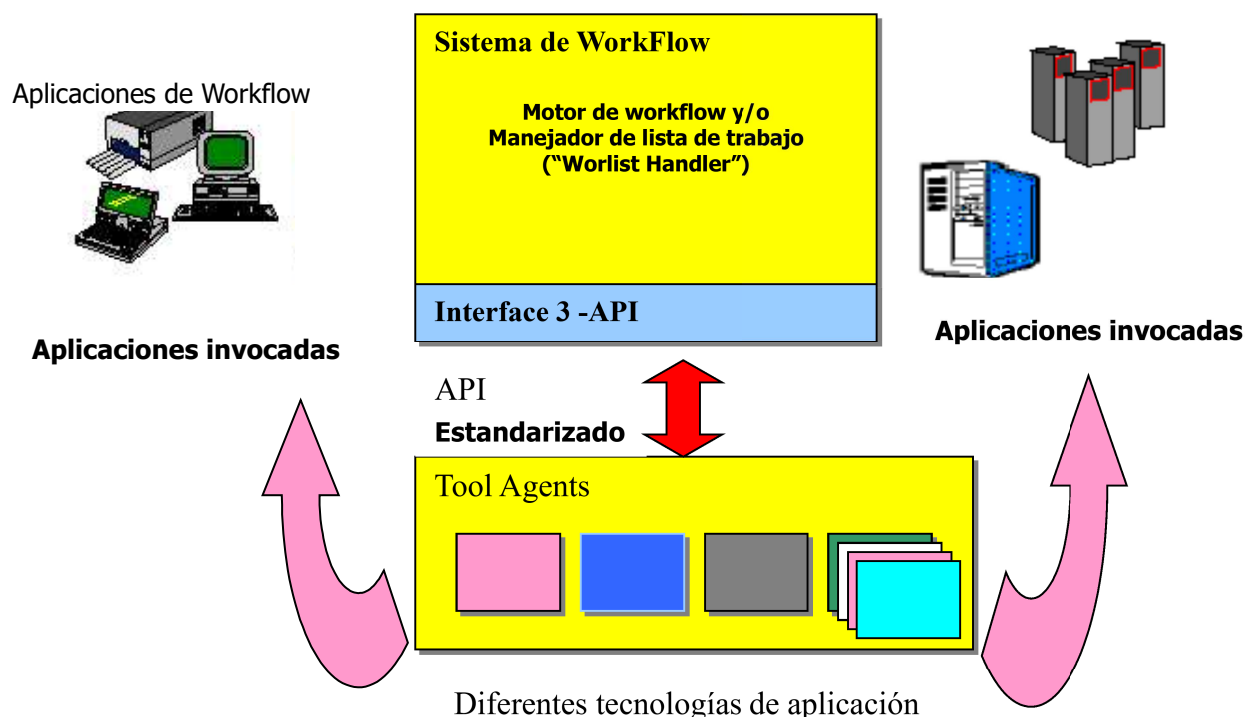


Figura 1: La localización de la interfase de invocación de aplicaciones

La arquitectura básica de los Tool Agents puede ser comparada por ejemplo con ODBC.

METODO	DESCRIPCION
WMTAConnect & WMTADisconnect	Estas APIs crean y terminan conexiones a interfases de Tool Agents. Estas APIs ya están definidas en la sección WAPI funciones de conexión
WMTAInvokeApplication	Obliga a un Tool Agent a empezar o activar una aplicación. El motor de workflow activa una aplicación específica asociada con un ítem de trabajo ("workitem") llamando a esta API del Tool Agent.
WMTARequestAppStatus	Permite al sistema de workflow por aplicaciones abiertas y su estado (corriendo, pendiente, etc). Define como el sistema de workflow tiene que chequear el estado de una aplicación y devolver al workflow datos relevantes de la aplicación.
WMTATerminateApp	Obliga al Tool Agent a terminar una aplicación.

4 INTERFASE 4 – INTEROPERABILIDAD

4.1 Definiciones

Motor de Workflow: El servicio de software que provee un ambiente de ejecución de instancias de flujos de trabajo (también llamados procesos).

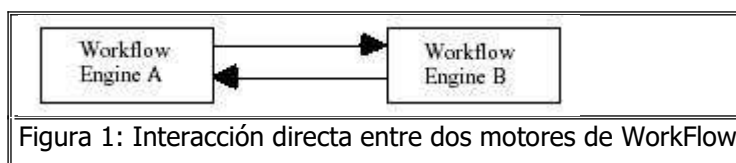
Workflow Enactment System(representación de los servicios de WorkFlow): Es un ambiente de ejecución en el cual la instanciación y activación de procesos ocurre utilizando uno o más motores de workflow, responsables de interpretar y activar parte o toda la definición de un proceso e interactuar con los recursos externos necesarios para procesar las actividades que integran los procesos.

Workflow Managment System (Sistema de Administración de WorkFlow): Sistema que define completamente, administra y ejecuta procesos, a través de la ejecución de piezas de software cuyo orden de ejecución es administrado por una representación computarizada de procesos o de flujo de trabajo. Un sistema de administración de WorkFlow consiste de uno o más "WorkFlow Enactment Services", donde cada uno de estos se compone de uno o más motores de WorkFlow.

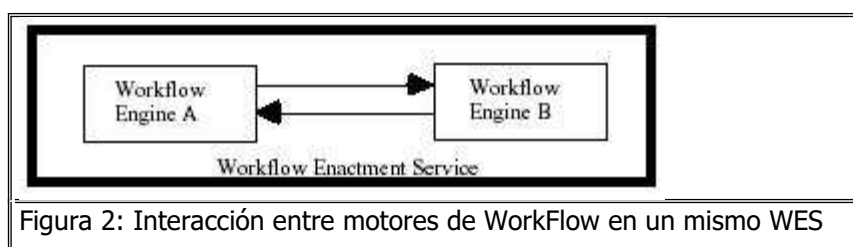
Interoperabilidad entre Motores de WorkFlows: Es la habilidad de dos o más motores de WorkFlow para comunicarse e interactuar de forma de coordinar entre ellos la ejecución de instancias de procesos de workflow.

En definitiva, estamos hablando de que dos motores de WF pueden interactuar entre sí, con distintos objetivos y en distintas situaciones, de lo que deducimos tres niveles de interacción:

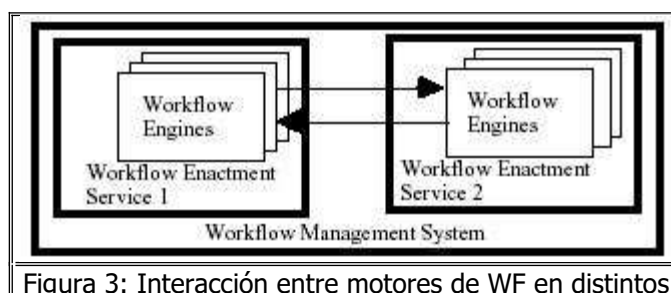
- Interacción directa entre dos motores de WF, que no necesariamente forman parte de un WES o un WFMS.



- Interacción entre dos motores de WF que conviven en un mismo WES.



- Interacción entre dos motores de WF que pertenecen a distintos WES, pero deben interactuar para satisfacer las necesidades de comunicación entre los WES que componen un mismo WMS.



WES, pero el mismo WMS

Dominios de WorkFlows: El modelo de referencia define dominios de workflow, los que contienen uno o más motores de workflow. Dos workflow interoperando deben compartir el mismo dominio, que define el contexto en el que la interoperabilidad toma lugar. Hay dos tipos básicos de dominio de Workflow:

- **Abiertos:** Describe el conjunto de motores de workflow, con los que un motor de workflow llega a interoperar, de forma directa o mediante la interoperabilidad con un motor intermedio (un "vecino" común).
- **Cerrados:** Hablamos de dominios cerrados cuando el dominio de un determinado motor, lo consideramos como el conjunto de motores con el que éste se comunica de forma directa, dejando de lado la interacción, vía otros motores intermedios.

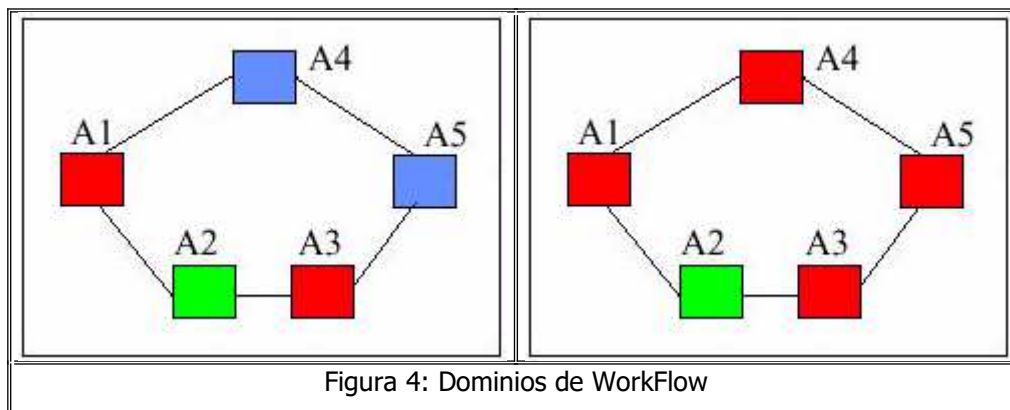


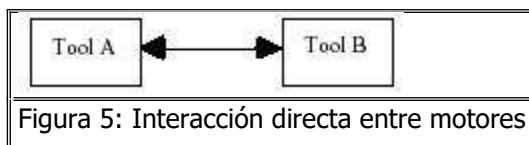
Figura 4: Dominios de Workflow

Si consideramos de dominio abierto de A2, estaría integrado por A1 y A3. Si consideramos en cambio, el dominio abierto de A2, obtendremos A1,A3,A4 y A5.

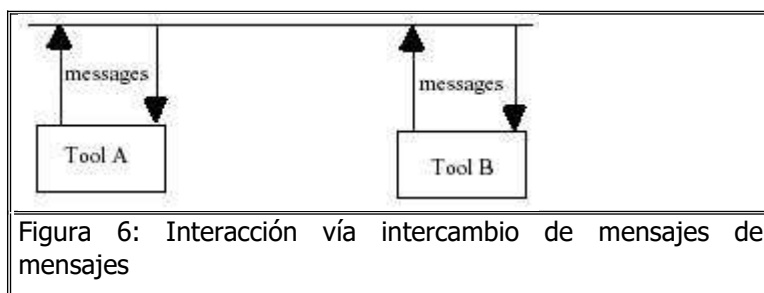
4.2 Estrategias de Interoperabilidad

La interoperabilidad entre herramientas de software, normalmente se hace con el objetivo de compartir datos y/o funcionalidades entre las mismas. A continuación, veremos algunas de las herramientas más comunes para la interoperabilidad entre piezas de software:

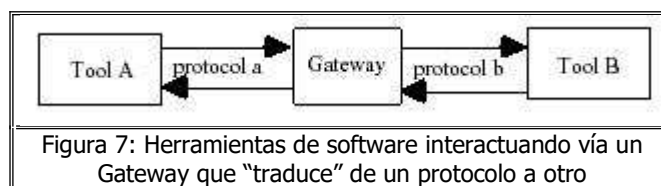
- **Interacción directa:** En este caso las piezas de software intercambian información en forma directa, por ejemplo, una es una dll y la otra conoce la definición de sus métodos por lo que puede llamar a sus funciones pasándole información en los parámetros de los métodos.



- **Intercambio de mensajes:** En estos casos existen algún tipo de mecanismo de intercambio de mensajes. Ejemplo de estos mecanismos podría ser mensajería por MSMQ, intercambio de correos, de mensajes por TCP/IP, broadcasting, etc. Alguna instancia manda un mensaje a otra, indicando que debe hacer y como. Ambas instancias deben conocer el "protocolo" de comunicación.



- **Técnica de "Bridging":** Esta técnica se usa cuando hay algún tipo de encapsulamiento o de gateway que administra la comunicación. Por ejemplo, los dos motores no tendrían por que conocer ni siquiera en qué equipo está corriendo el otro.



- **Utilización de un repositorio común:** En esta técnica la interoperabilidad se da por el intercambio de información a través de un repositorio común al que ambas herramientas tienen acceso. Por tanto, cada una puede leer información que la otra inserte en el repositorio.

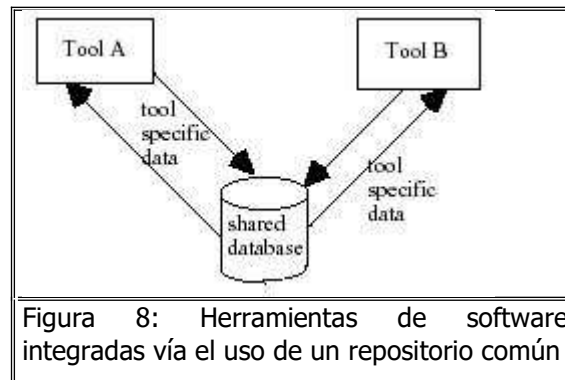


Figura 8: Herramientas de software integradas vía el uso de un repositorio común

4.3 Niveles de Interoperabilidad

La WfMC identifica 8 niveles de interoperabilidad. Los niveles de interoperabilidad se distinguen por la arquitectura y consecuentemente, por las características de implementación del motor de Workflow, que son necesarias para satisfacer cada uno de ellos..

Nivel 1:

No hay interoperabilidad.

Nivel 2 (Coexistencia):

Esta es una aproximación no estándar a la interoperabilidad de productos de Workflow. Es, sobre todo, una apuesta de los desarrolladores de Workflow para mejorar la performance de sus productos en múltiples plataformas. El efecto es un mayor número de productos de workflow, que pueden coexistir en distintas plataformas. Este nivel es caracterizado por productos de workflow compartiendo el mismo ambiente de ejecución. Esto no implica ninguna interacción entre los productos, pero permite a las empresas satisfacer sus necesidades mediante un conjunto de aplicaciones disponibles.

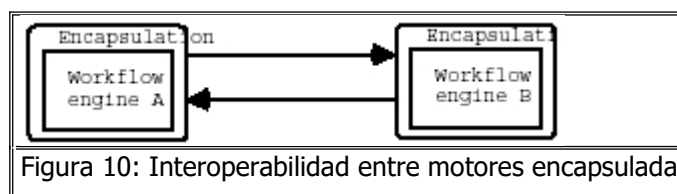
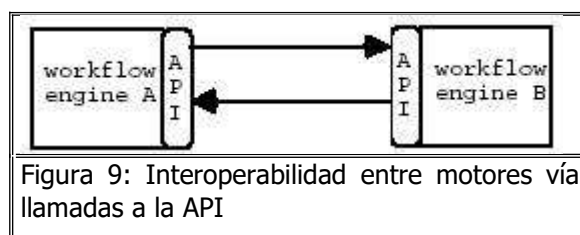
Nivel 3 (Unique Gateway):

Este nivel es caracterizado por productos de WF trabajando juntos, usando algún mecanismo de interoperabilidad que satisfaga:

- Ruteo de operaciones entre motores e instancias.
- Distribución de datos relevantes del Workflow.
- Distribución de datos de aplicaciones del Workflow.

Nivel 3.a (Common Gateway API): Este nivel se caracteriza por productos de workflow trabajando juntos, usando gateways que compartan una API común. Este nivel acarrea la necesidad de que la implementación de las operaciones soportadas por los distintos WF sean normalizadas para generar un subconjunto de las mismas que se utilicen como estándar, pero no excluye la posibilidad de implementar supraconjuntos.

Nivel 4 (Subconjunto Común limitado de la API): Los productos de WF en este nivel, comparten una API común que les permite interactuar moviendo y administrando el trabajo entre ellos. Implementar este nivel de interoperabilidad, requiere un conjunto central de funciones de la API implementadas por los motores de WF en cuestión. Los modelos de implementación para este nivel, son actualmente sumamente simples y la interacción se hace por uso de API's o encapsulamiento.



Como es obvio que no podemos exigir que todos los WF manejan la información de la misma forma. Lo que se hace, en general, es definir un formato neutral en el que se administra la información. Cada WF debe saber leer (y escribir) desde (y en) ese formato.

Nivel 5 (API Completamente compartida): Es un paso más en los niveles de interoperabilidad, que consiste en que los distintos motores de un WF implementen toda la API, y no una parte. Exige que todos los WF implementen toda la API, lo que no excluye que cada WF brinde funciones extra que sean aplicables al dominio particular.

Como primer paso se define la API y luego cada WF debe mapear sus propias funciones a las de la API. La implementación de estos mapeos puede requerir un mecanismo similar al del nivel 4.

Nivel 6 (Formatos de Definición compartidos): Este nivel se caracteriza por distintos productos de WF teniendo un formato compartido para la definición de procesos, que cubra las definiciones de los flujos, los derechos de acceso de los usuarios y el mantenimiento de los recursos de los sistemas de WF. La consecuencia natural de estas premisas, es que una organización (sea una o varias compañías) hacen una única definición de un proceso, en un lenguaje común a varios motores de WF y luego estos procesos se satisfacen independientemente de que motor los ejecute realmente. Una definición en este sentido que se precie de realista, no puede asumir que todos los sistemas de WF implementarán todas las operaciones, por lo que se debe definir un conjunto mínimo, tal que todos los sistemas de WF que quieran alcanzar este nivel de interoperabilidad, deben implementar. Todas las operaciones por sobre este estándar mínimo, serán consideradas potencialidades extra del motor de WF en cuestión. Para satisfacer este nivel de interoperabilidad, un sistema de WF debe soportar:

- La definición de un conjunto genérico de funcionalidades.
- La definición de perfiles operacionales para diferentes clases de usuarios del sistema de Workflow, para identificar los que pueden ser utilizados para satisfacer distintas funcionalidades.

La WPMC tiene definido un lenguaje de definición de procesos (WPD), para tomar el primer paso. Las funcionalidades adicionales se pueden ofrecer utilizando definiciones extendidas del lenguaje. En este momento la WPMC, maneja el lenguaje XPD (WPD extensible).

Estos conceptos, permiten avanzar un paso más en la interoperabilidad entre sistemas de WF. Ese paso es brindar la posibilidad de que distintos motores de WF, puedan ejecutar no solo de forma indistinta un proceso, sino que pueden ejecutar distintas etapas o actividades de un mismo proceso.

Internamente, cada motor de WF puede administrar los procesos como mejor le parezca, al igual que el editor de procesos, pero deben implementar dos interfaces:

- **Importar:** Del proceso definido en el lenguaje común y expresarlo en el lenguaje particular.
- **Exportar:** Un proceso de su lenguaje particular a un stream de caracteres que satisfaga el lenguaje común de definición de procesos.

Nivel 7 (Compatibilidad de Protocolos): Este nivel asume que toda la API de comunicación cliente/servidor, incluyendo la transmisión de definiciones de procesos, las transacciones, la recuperación de procesos suspendidos, etc., están estandarizadas. En este caso la interoperabilidad puede ir un paso más allá y alcanzar niveles como que el cliente hace solicitudes a un servidor virtual, el cual redirige a distintos motores la solicitud, sin que importe mayormente cual la recibe. Esta forma de trabajar es al estilo de granja de servidores.

Nivel 8 (Utilidades comunes de "Look And Feel"): En este nivel de interoperabilidad, los distintos productos de WF deben respetar estándares de interfaces de usuario, por asociación. Es decir, las interfaces de usuarios deben seguir prácticamente la misma distribución, para lograr que el impacto para el usuario de pasar de un sistema a otro sea prácticamente nulo.

4.4 Modelos de interoperabilidad

Los modelos de interoperabilidad podrían clasificarse rápidamente en cuatro grupos:

- Procesos encadenados
- Subprocesos sincrónicos anidados
- Subprocesos sincronizados por eventos
- Subprocesos anidados con sincronización diferida

Procesos encadenados:

El proceso se comienza ejecutando en un motor de WF A. Desde éste, se instancia un subproceso en el motor de WF B. Luego de la instanciación cada motor sigue con su actividad de forma independiente.

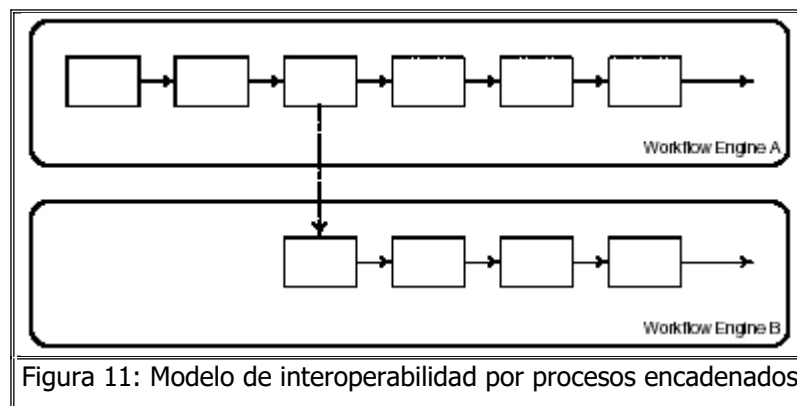


Figura 11: Modelo de interoperabilidad por procesos encadenados

Subprocesos sincrónicos anidados:

Este modelo asume que una instancia corriendo en un primer motor de WF, invoca a un subproceso que se ejecuta en un segundo motor.

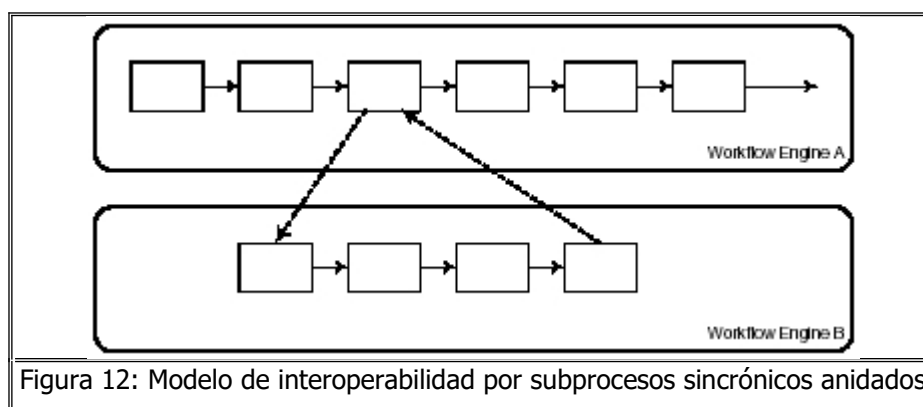


Figura 12: Modelo de interoperabilidad por subprocesos sincrónicos anidados

La actividad que llama al subproceso que corre en el segundo motor, queda suspendida hasta que el subproceso termina y, de alguna manera, la sincronización se logra por notificación (mediante el cambio del valor de un determinado atributo en el estado del subproceso).

Subprocesos sincronizados por eventos:

Dos procesos corren en paralelo en distintos motores y, cada tanto, hay actividades de sincronización, como se ve en la figura, en forma de evento.

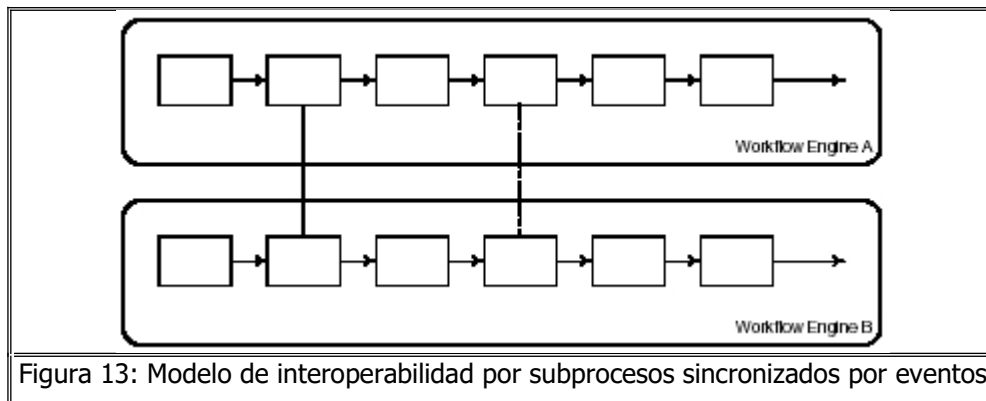


Figura 13: Modelo de interoperabilidad por subprocesos sincronizados por eventos

Subprocesos anidados con sincronización diferida:

Este modelo permite a un proceso corriendo en un motor de WF, crear otros procesos en otro motor, con solo conocer su definición. El proceso que hace la invocación sigue adelante, en paralelo con el invocado, hasta un punto donde necesita un resultado del otro motor. En este momento, hace un pooling esperando por la respuesta hasta que llegue.

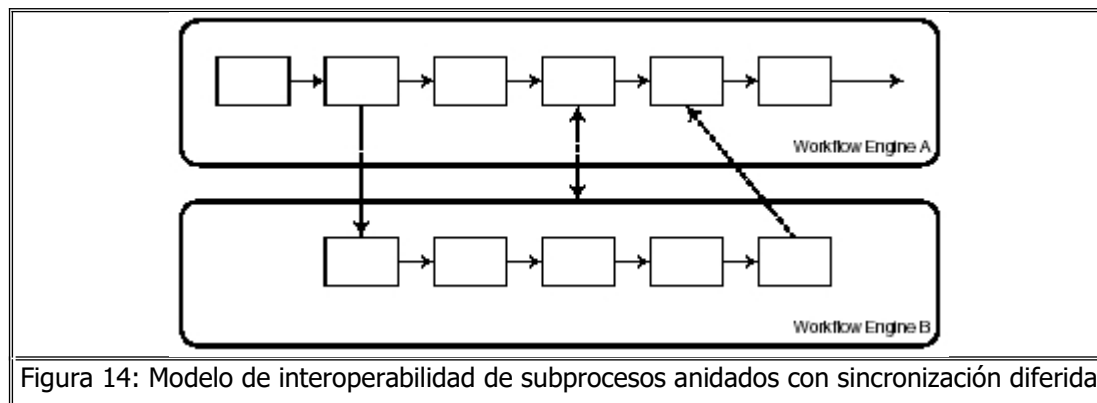


Figura 14: Modelo de interoperabilidad de subprocesos anidados con sincronización diferida

Iniciar un subproceso encadenado:

Este escenario involucra al motor de WF que invoca (el que corre el proceso padre), que crea e inicia una nueva instancia de un procesos que se corre en otro motor de WF. El motor que invoca, no espera por el subproceso que instancia. Luego de la interacción, el primero sigue con su actividad libremente, mientras que el segundo continúa con las siguientes actividades del proceso.

Ahora definiremos los requerimientos mínimos para que este tipo de interacción tenga lugar:

El primer motor de WF (llamémoslo A) debe:

- Tener la posibilidad de seleccionar una definición de proceso manejada por B o accesible a B.
- Pasar al motor de WF B la "relevant data" para poder instanciar la definición de proceso seleccionada.
- Hacer un request al motor B para que active la definición de proceso seleccionada.

La transferencia de datos propios de las aplicaciones entre los motores está fuera del alcance de este documento. Es claro que los workflow deben compartir la definición de los procesos de alguna forma. Las opciones son:

- La definición de los procesos a ejecutar es pasada de un motor a otro.
- La ubicación de las definiciones de los procesos a ejecutar debe ser conocida y accesible por ambos motores.

Este estándar asume que, si es necesario pasar la definición de subprocesos entre los motores, esta transferencia se hace en la llamada. A la hora de establecer comunicación entre los motores tenemos dos opciones. Podemos hacerlo de forma sincrónica (por ejemplo, vía conexión TCP/IP) o asincrónica (por ejemplo, vía e-mail).

Motor		Operación
A		Crear una instancia de proceso
B		Responder a la creación de la instancia de proceso
A		Setear los atributos de la instancia de proceso
B		Responder al seteo de atributos
B		Obtener atributos de la instancia de proceso
A		Responder a la obtención de atributos de la instancia de proceso
A		Cambiar el estado de la instancia del proceso
B		Responder al cambio de estado de la instancia de proceso
A		Abandonar la instancia de proceso
B		Responder al mensaje de abandono de la instancia de proceso
Operaciones necesarias para iniciar un subproceso encadenado		

Como se observa en la tabla, el diálogo entre los dos motores de WF se basa en la noción de pares de mensajes de pedido/respuesta donde las respuestas son indicadores de éxito/fracaso de la llamada.

Iniciar un subproceso que completa un paso de un proceso:

En este escenario, la instancia del proceso padre espera hasta que el hijo complete su parte del proceso. Para soportar este tipo de escenarios, debemos proveer un mecanismo por el cual el motor "B" avise al motor "A" que terminó su parte del proceso para que este continúe.

En la siguiente tabla observamos las operaciones necesarias para la comunicación

Motor		Operación
A		Crear una Instancia de un proceso
B		Responder a la creación de la instancia de proceso
A		Setear los atributos de la instancia de proceso
B		Responder al seteo de atributos
B		Obtener los atributos de la instancia de proceso
A		Responder a la solicitud de los atributos de la instancia
A		Cambiar el estado de la instancia de proceso
B		Responder al cambio de estado
B		Atributos cambiados de la instancia de proceso
A		Responder a los cambios en los atributos de la instancia de proceso
A		Obtener los atributos de la instancia de proceso
B		Responder a la obtención de atributos de la instancia de proceso
A		Desligarse de la instancia de proceso

B	Responder al desligar de la instancia de proceso
Operaciones necesarias para que un subprocesso complete un paso de un proceso original	

Administración de Procesos:

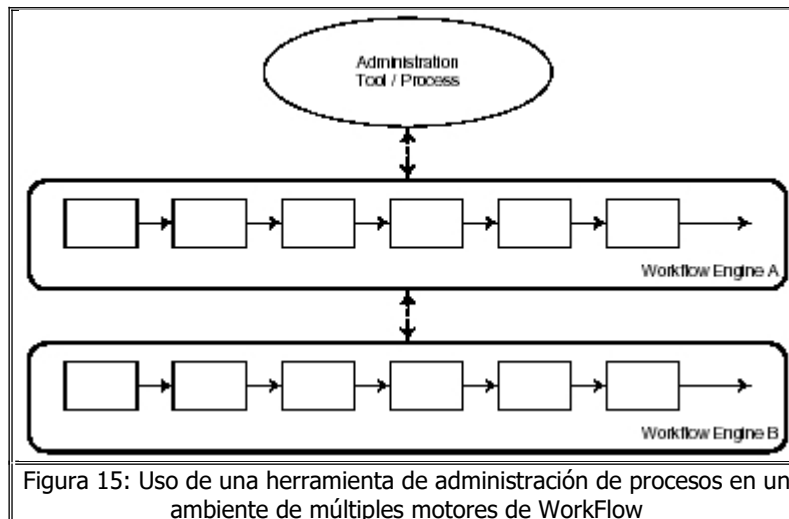


Figura 15: Uso de una herramienta de administración de procesos en un ambiente de múltiples motores de Workflow

En la figura anterior, vemos representadas por flechas las interacciones necesarias para trabajar en un entorno con un administrador de procesos. Si tomamos como referencia el motor llamado "A" en el ejemplo, que debe implementar dos métodos extra para soportar el uso de herramientas de administración de procesos en un ambiente de múltiples motores de Workflow:

- Listar las instancias de procesos
- Obtener los cambios de estado de las instancias de proceso

Estado de los procesos:

La siguiente imagen muestra los posibles estados de un proceso.

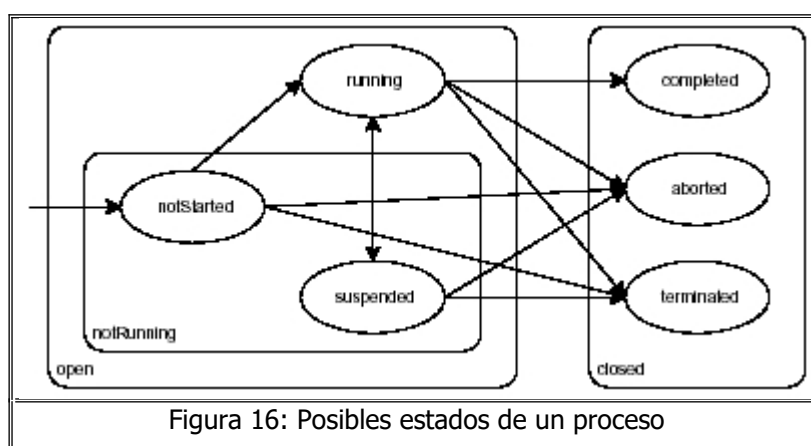


Figura 16: Posibles estados de un proceso

Básicamente un proceso puede estar abierto o cerrado. Si está abierto puede estar corriendo o no. En caso de no estar corriendo hay dos opciones, que no se haya iniciado aun, o que esté momentáneamente suspendido. En el caso de estar cerrado, puede estar completado, abortado o terminado.

En el diagrama anterior se muestran también las transiciones posibles entre estados.

5 REFERENCIAS DE INTERES:

- David Hollingsworth - TC-1003 Workflow Reference Model – WfMC – 1995
<http://www.wfmc.org/standards/docs.htm>. - 20/5/2004
- WfMC – TC-1011 Terminology & Glosary - 1999
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- David Hollingsworth WfMC – TC-1023 Common Object Model - WfMC - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC Work Group 1 – TC-1016-X Process Definition Interchange - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC Work Group 1 – TC-1016-P Process Definition Interchange ,Process Model - 1999
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC Work Group 1 – TC- 1016-O Process Definition Interchange – Organizational Model - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC-1025 Process Definition Interface – XML Process Definition Language - 2002
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC – 1009 Aplication Programming Interface (WAPI) – Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- [10] – WfMC – TC- 1013 Aplication Programming Interface – Naming Conventions - 1997
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC – 1012 Interoperability – Abstract Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC -1015 Audit Data Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – Workflow Handbook 2003 - 2003
<http://www.wfmc.org/information/handbook03.htm> - 20/5/2004
- WfMC – Workflow Handbook 2002 - 2002
<http://www.wfmc.org/information/handbook02.htm> - 20/5/2004
- WfMC – Workflow Handbook 2001 - 2001
<http://www.wfmc.org/information/handbook01.htm> - 20/5/2004

PROYECTO DE GRADO – SISTEMAS DE WORKFLOW.

Javier Apaulaza, José Casarini, Rodrigo Recoba

Tutor: Guillermo Calderón
Usuario Responsable: Daniel Gómez

Instituto de Computación
Facultad de Ingeniería
Universidad de la República Oriental del Uruguay.

Marzo 2004

Resumen.

En este apéndice, se presentan resúmenes sobre la plataforma y lenguaje utilizados. También se presentan diferentes herramientas estudiadas y utilizadas para la realización del prototipo. Dichos resúmenes son de nivel técnico e intentan explicar en forma detallada el marco en el cual se construyó el prototipo, y las funcionalidades que el framework brinda a la hora de su diseño e implementación.

ÍNDICE

1	INTRODUCCIÓN A .NET.....	3
1.1	Common Language Runtime (CLR).....	3
1.2	Microsoft Intermediate Language (MSIL).....	5
1.3	Metadatos.....	6
1.4	Ensamblados.....	7
1.5	Librería de clase base (BCL).....	8
1.6	Concepto de espacio de nombres.....	8
1.7	Common Type System (CTS).....	9
1.8	Common Language Specification (CLS).....	10
2	INTRODUCCIÓN A C#.....	11
2.1	Origen y necesidad de un nuevo lenguaje.....	11
2.2	Características de C#.....	11
3	INTRODUCCIÓN A XML.....	16
3.1	En memoria vs. ForwardOnly.....	17
3.2	Cargando el documento en un formato ForwardOnly.....	18
3.3	DOM.....	20
3.4	XPath.....	22
3.5	¿Cómo validar un documento XML?.....	25
4	INTRODUCCIÓN ADO.NET.....	28
4.1	ADO.NET en la plataforma .NET Framework:.....	28
4.2	ADO.NET.....	29
4.3	Objetivos del diseño de ADO.NET.....	32
4.4	DataSet de ADO.NET.....	35
4.5	Adaptadores de datos de .NET.....	38
4.6	LOS ESPACIOS DE NOMBRES DE ADO.NET.....	40
4.7	Conceptos interesantes al enfrentarse a ADO.NET.....	41
5	INTRODUCCIÓN A ASP.NET.....	43
6	SERVICIOS EN .NET.....	47
6.1	Aplicaciones de servicios y otras aplicaciones de Visual Studio .NET.....	47
6.2	Ciclo de vida de los servicios.....	47
6.3	Tipos de servicios.....	48
6.4	Los servicios y el componente ServiceController.....	48
6.5	Implementar e instalar servicios.....	48
7	Referencias de interés.....	52

1 INTRODUCCIÓN A .NET

Microsoft.NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. Esta es la llamada **plataforma .NET**, y a los servicios antes comentados se les denomina **servicios Web**.

Para crear aplicaciones para la plataforma .NET, tanto servicios Web como aplicaciones tradicionales (aplicaciones de consola, aplicaciones de ventanas, servicios de Windows NT, etc.), Microsoft ha publicado el denominado kit de desarrollo de software conocido como **.NET Framework SDK**, que incluye las herramientas necesarias tanto para su desarrollo como para su distribución y ejecución y **Visual Studio.NET**, que permite hacer todo lo anterior desde una interfaz visual basada en ventanas.

El concepto de Microsoft.NET también incluye al conjunto de nuevas aplicaciones que Microsoft y terceros han (o están) desarrollando para ser utilizadas en la plataforma .NET.

1.1 Common Language Runtime (CLR)

El **Common Language Runtime (CLR)** es el núcleo de la plataforma .NET. Es el motor encargado de gestionar la ejecución de las aplicaciones para ella desarrolladas y a las que ofrece numerosos servicios que simplifican su desarrollo y favorecen su fiabilidad y seguridad. Las principales características y servicios que ofrece el CLR son:

- **Modelo de programación consistente:** A todos los servicios y facilidades ofrecidos por el CLR se accede de la misma forma: a través de un modelo de programación orientado a objetos. Esto es una diferencia importante respecto al modo de acceso a los servicios ofrecidos por los algunos sistemas operativos actuales (por ejemplo, los de la familia Windows), en los que a algunos servicios se les accede a través de llamadas a funciones globales definidas en DLLs y a otros a través de objetos (objetos COM en el caso de la familia Windows)
- **Modelo de programación sencillo:** Con el CLR desaparecen muchos elementos complejos incluidos en los sistemas operativos actuales (registro de Windows, GUIDs, HRESULTS, IUnknown, etc.) El CLR no es que abstraiga al programador de estos conceptos, sino que son conceptos que no existen en la plataforma .NET
- **Eliminación del "infierno de las DLLs":** En la plataforma .NET desaparece el problema conocido como "infierno de las DLLs" que se da en los sistemas operativos actuales de la familia Windows, problema que consiste en que al sustituirse versiones viejas de DLLs compartidas por versiones nuevas puede que aplicaciones que fueron diseñadas para ser ejecutadas usando las viejas dejen de funcionar si las nuevas no son 100% compatibles con las anteriores. En la plataforma .NET las versiones nuevas de las DLLs pueden coexistir con las viejas, de modo que las aplicaciones diseñadas para ejecutarse usando las viejas podrán seguir usándolas tras instalación de las nuevas. Esto, obviamente, simplifica mucho la instalación y desinstalación de software.
- **Ejecución multiplataforma:** El CLR actúa como una máquina virtual, encargándose de ejecutar las aplicaciones diseñadas para la plataforma .NET. Es decir, cualquier plataforma para la que exista una versión del CLR podrá ejecutar cualquier aplicación .NET. Microsoft ha desarrollado versiones del CLR para la mayoría de las versiones de Windows: Windows 95, Windows 98, Windows ME, Windows NT 4.0, Windows 2000, Windows XP y Windows CE (que puede ser usado en CPUs que no sean de la familia x86) Por otro lado Microsoft ha firmado un acuerdo con Corel para portar el CLR a Linux y también hay terceros que están desarrollando de manera independiente versiones de libre distribución del CLR para Linux. Asimismo, dado que la arquitectura del CLR está totalmente abierta, es posible que en el futuro se diseñen versiones del mismo para otros sistemas operativos.

- **Integración de lenguajes:** Desde cualquier lenguaje para el que exista un compilador que genere código para la plataforma .NET es posible utilizar código generado para la misma usando cualquier otro lenguaje tal y como si de código escrito usando el primero se tratase. Microsoft ha desarrollado un compilador de C# que genera código de este tipo, así como versiones de sus compiladores de Visual Basic (Visual Basic.NET) y C++ (C++ con extensiones gestionadas) que también lo generan y una versión del intérprete de JScript (JScript.NET) que puede interpretarlo. La integración de lenguajes esta que es posible escribir una clase en C# que herede de otra escrita en Visual Basic.NET que, a su vez, herede de otra escrita en C++ con extensiones gestionadas.
- **Gestión de memoria:** El CLR incluye un **recolector de basura** que evita que el programador tenga que tener en cuenta cuándo ha de destruir los objetos que dejen de serle útiles. Este recolector es una aplicación que se activa cuando se quiere crear algún objeto nuevo y se detecta que no queda memoria libre para hacerlo, caso en que el recolector recorre la memoria dinámica asociada a la aplicación, detecta qué objetos hay en ella que no puedan ser accedidos por el código de la aplicación, y los elimina para limpiar la memoria de "objetos basura" y permitir la creación de otros nuevos. Gracias a este recolector se evitan errores de programación muy comunes como intentos de borrado de objetos ya borrados, agotamiento de memoria por olvido de eliminación de objetos inútiles o solicitud de acceso a miembros de objetos ya destruidos.
- **Seguridad de tipos:** El CLR facilita la detección de errores de programación difíciles de localizar comprobando que toda conversión de tipos que se realice durante la ejecución de una aplicación .NET se haga de modo que los tipos origen y destino sean compatibles.
- **Aislamiento de procesos:** El CLR asegura que desde código perteneciente a un determinado proceso no se pueda acceder a código o datos pertenecientes a otro, lo que evita errores de programación muy frecuentes e impide que unos procesos puedan atacar a otros. Esto se consigue gracias al sistema de seguridad de tipos antes comentado, pues evita que se pueda convertir un objeto a un tipo de mayor tamaño que el suyo propio, ya que al tratarlo como un objeto de mayor tamaño podría accederse a espacios en memoria ajenos a él que podrían pertenecer a otro proceso. También se consigue gracias a que no se permite acceder a posiciones arbitrarias de memoria.
- **Tratamiento de excepciones:** En el CLR todos los errores que se puedan producir durante la ejecución de una aplicación se propagan de igual manera: mediante excepciones. Esto es muy diferente a como se venía haciendo en los sistemas Windows hasta la aparición de la plataforma .NET, donde ciertos errores se transmitían mediante códigos de error en formato Win32, otros mediante HRESULTs y otros mediante excepciones.

El CLR permite que excepciones lanzadas desde código para .NET escrito en un cierto lenguaje se puedan capturar en código escrito usando otro lenguaje, e incluye mecanismos de depuración que pueden saltar desde código escrito para .NET en un determinado lenguaje a código escrito en cualquier otro. Por ejemplo, se puede recorrer la pila de llamadas de una excepción aunque ésta incluya métodos definidos en otros módulos usando otros lenguajes.

- **Soporte multihilo:** El CLR es capaz de trabajar con aplicaciones divididas en múltiples hilos de ejecución que pueden ir evolucionando por separado en paralelo o intercalándose, según el número de procesadores de la máquina sobre la que se ejecuten. Las aplicaciones pueden lanzar nuevos hilos, destruirlos, suspenderlos por un tiempo o hasta que les llegue una notificación, enviarles notificaciones, sincronizarlos, etc.
- **Distribución transparente:** El CLR ofrece la infraestructura necesaria para crear objetos remotos y acceder a ellos de manera completamente transparente a su localización real, tal y como si se encontrasen en la máquina que los utiliza.

- **Seguridad avanzada:** El CLR proporciona mecanismos para restringir la ejecución de ciertos códigos o los permisos asignados a los mismos según su procedencia o el usuario que los ejecute. Es decir, puede no darse el mismo nivel de confianza a código procedente de Internet que a código instalado localmente o procedente de una red local; puede no darse los mismos permisos a código procedente de un determinado fabricante que a código de otro; y puede no darse los mismos permisos a un mismo código según el usuario que lo esté ejecutando o según el rol que éste desempeñe. Esto permite asegurar al administrador de un sistema que el código que se esté ejecutando no pueda poner en peligro la integridad de sus archivos, la del registro de Windows, etc.
- **Interoperabilidad con código antiguo:** El CLR incorpora los mecanismos necesarios para poder acceder desde código escrito para la plataforma .NET a código escrito previamente a la aparición de la misma y, por tanto, no preparado para ser ejecutado dentro de ella. Estos mecanismos permiten tanto el acceso a objetos COM como el acceso a funciones sueltas de DLLs preexistentes (como la API Win32)

Como se puede deducir de las características comentadas, el CLR lo que hace es gestionar la ejecución de las aplicaciones diseñadas para la plataforma .NET. Por esta razón, al código de estas aplicaciones se le suele llamar **código gestionado**, y al código no escrito para ser ejecutado directamente en la plataforma .NET se le suele llamar **código no gestionado**.

1.2 Microsoft Intermediate Language (MSIL)

Todos los compiladores que generan código para la plataforma .NET no generan código máquina para CPUs x86 ni para ningún otro tipo de CPU concreta, sino que generan código escrito en el lenguaje intermedio conocido como Microsoft Intermediate Language (MSIL). El CLR da a las aplicaciones la sensación de que se están ejecutando sobre una máquina virtual, y precisamente MSIL es el código máquina de esa máquina virtual. Es decir, MSIL es el único código que es capaz de interpretar el CLR, y por tanto cuando se dice que un compilador genera código para la plataforma .NET lo que se está diciendo es que genera MSIL.

MSIL ha sido creado por Microsoft tras consultar a numerosos especialistas en la escritura de compiladores y lenguajes tanto del mundo académico como empresarial. Es un lenguaje de un nivel de abstracción mucho más alto que el de la mayoría de los códigos máquina de las CPUs existentes, e incluye instrucciones que permiten trabajar directamente con objetos (crearlos, destruirlos, inicializarlos, llamar a métodos virtuales, etc.), tablas y excepciones (lanzarlas, capturarlas y tratarlas)

Ya se comentó que el compilador de C# compila directamente el código fuente a MSIL, que Microsoft ha desarrollado nuevas versiones de sus lenguajes Visual Basic (Visual Basic.NET) y C++ (C++ con extensiones gestionadas) cuyos compiladores generan MSIL, y que ha desarrollado un intérprete de JScript (JScript.NET) que genera código MSIL. Pues bien, también hay numerosos terceros que han anunciado estar realizando versiones para la plataforma .NET de otros lenguajes como APL, CAML, Cobol, Eiffel, Fortran, Haskell, Java (J#), Mercury, ML, Mondrian, Oberon, Oz, Pascal, Perl, Python, RPG, Scheme y Smalltalk.

La principal ventaja del MSIL es que facilita la ejecución multiplataforma y la integración entre lenguajes al ser independiente de la CPU y proporcionar un formato común para el código máquina generado por todos los compiladores que generen código para .NET. Sin embargo, dado que las CPUs no pueden ejecutar directamente MSIL, antes de ejecutarlo habrá que convertirlo al código nativo de la CPU sobre la que se vaya a ejecutar. De esto se encarga un componente del CLR conocido como compilador JIT (Just-In-Time) o jitter que va convirtiendo dinámicamente el código MSIL a ejecutar en código nativo según sea necesario. Este jitter se distribuye en tres versiones:

- **jitter normal:** Es el que se suele usar por defecto, y sólo compila el código MSIL a código nativo a medida que va siendo necesario, pues así se ahorra tiempo y memoria al evitarse tener que compilar innecesariamente código que nunca se ejecute. Para conseguir esto, el cargador de clases del CLR sustituye inicialmente las llamadas a métodos de las nuevas clases que vaya cargando por llamadas a funciones auxiliares (stubs) que se encarguen de compilar el verdadero código del método. Una vez compilado, la llamada al stub es sustituida por una llamada directa al código ya compilado, con lo que posteriores llamadas al mismo no necesitarán compilación.
- **jitter económico:** Funciona de forma similar al jitter normal solo que no realiza ninguna optimización de código al compilar sino que traduce cada instrucción MSIL por su equivalente en el código máquina sobre la que se ejecute. Está especialmente pensado para ser usado en dispositivos empotrados que dispongan de poca potencia de CPU y poca memoria, pues aunque genere código más ineficiente es menor el tiempo y memoria que necesita para compilar. Es más, para ahorrar memoria este jitter puede descargar código ya compilado que lleve cierto tiempo sin ejecutarse y sustituirlo de nuevo por el stub apropiado. Por estas razones, este es el jitter usado por defecto en Windows CE, sistema operativo que se suele incluir en los dispositivos empotrados antes mencionados.

Otra utilidad del jitter económico es que facilita la adaptación de la plataforma .NET a nuevos sistemas porque es mucho más sencillo de implementar que el normal. De este modo, gracias a él es posible desarrollar rápidamente una versión del CLR que pueda ejecutar aplicaciones gestionadas aunque sea de una forma poco eficiente, y una vez desarrollada es posible centrarse en desarrollar el jitter normal para optimizar la ejecución de las mismas.

- **prejitter:** Se distribuye como una aplicación en línea de comandos llamada **ngen.exe** mediante la que es posible compilar completamente cualquier ejecutable o librería (cualquier ensamblado en general, aunque este concepto se verá más adelante) que contenga código gestionado y convertirlo a código nativo, de modo que posteriores ejecuciones del mismo se harán usando esta versión ya compilada y no se perderá tiempo en hacer la compilación dinámica.

La actuación de un jitter durante la ejecución de una aplicación gestionada puede dar la sensación de hacer que ésta se ejecute más lentamente debido a que ha de invertirse tiempo en las compilaciones dinámicas. Esto es cierto, pero hay que tener en cuenta que es una solución mucho más eficiente que la usada en otras plataformas como Java, ya que en .NET cada código no es interpretado cada vez que se ejecuta sino que sólo es compilado la primera vez que se llama al método al que pertenece. Es más, el hecho de que la compilación se realice dinámicamente permite que el jitter tenga acceso a mucha más información sobre la máquina en que se ejecutará la aplicación del que tendría cualquier compilador tradicional, con lo que puede optimizar el código para ella generado (por ejemplo, usando las instrucciones especiales del Pentium III si la máquina las admite, usando registros extra, incluyendo código *inline*, etc.) Además, como el recolector de basura de .NET mantiene siempre compactada la memoria dinámica las reservas de memoria se harán más rápido, sobre todo en aplicaciones que no agoten la memoria y, por tanto, no necesiten de una recolección de basura. Por estas razones, los ingenieros de Microsoft piensan que futuras versiones de sus jitters podrán incluso conseguir que el código gestionado se ejecute más rápido que el no gestionado.

1.3 Metadatos

En la plataforma .NET se distinguen dos tipos de **módulos** de código compilado: **ejecutables** (extensión **.exe**) y **librerías de enlace dinámico** (extensión **.dll** generalmente) Ambos son ficheros que contienen definiciones de tipos de datos, y la diferencia entre ellos es que sólo los primeros disponen de un método especial que sirve de punto de entrada a partir del que es posible ejecutar el código que contienen haciendo una llamada desde la línea de comandos del sistema operativo. A ambos tipos de módulos se les suele llamar **ejecutables portables** (PE), ya que su código puede ejecutarse en cualquiera de los diferentes sistemas operativos de la familia Windows para los que existe alguna versión del CLR.

El contenido de un módulo no sólo MSIL, sino que también consta de otras dos áreas muy importantes: la cabecera de CLR y los metadatos:

- La **cabecera de CLR** es un pequeño bloque de información que indica que se trata de un módulo gestionado e indica la versión del CLR que necesita, cuál es su firma digital, cuál es su punto de entrada (si es un ejecutable), etc.

Los **metadatos** son un conjunto de datos organizados en forma de tablas que almacenan información sobre los tipos definidos en el módulo, los miembros de éstos y sobre cuáles son los tipos externos al módulo a los que se les referencia en el módulo.

1.4 Ensamblados

Un **ensamblado** es una agrupación lógica de uno o más módulos o ficheros de recursos (ficheros .GIF, .HTML, etc.) que se engloban bajo un nombre común. Un programa puede acceder a información o código almacenado en un ensamblado sin tener porqué sabe cuál es el fichero en concreto donde se encuentran, por lo que los ensamblados nos permiten abstraernos de la ubicación física del código que ejecutemos o de los recursos que usemos. Por ejemplo, podemos incluir todos los tipos de una aplicación en un mismo ensamblado pero colocando los más frecuentemente usados en un cierto módulo y los menos usados en otro, de modo que sólo se descarguen de Internet los últimos si es que se van a usar.

Todo ensamblado contiene un **manifiesto**, que son metadatos con información sobre las características del ensamblado. Este manifiesto puede almacenarse cualquiera de los módulos que formen el ensamblado o en uno específicamente creado para ello, caso éste último necesario cuando es un **ensamblado satélite** (sólo contiene recursos)

Para asegurar que no se haya alterado la información de ningún ensamblado se usa el criptosistema de clave pública RSA. Lo que se hace es calcular el código de dispersión SHA-1 del módulo que contenga el manifiesto e incluir tanto este valor cifrado con RSA (**firma digital**) como la clave pública necesaria para descifrarlo en algún lugar del módulo que se indicará en la cabecera de CLR. Cada vez que se vaya a cargar en memoria el ensamblado se calculará su valor de dispersión de nuevo y se comprobará que es igual al resultado de descifrar el original usando su clave pública. Si no fuese así se detectaría que se ha adulterado su contenido.

Para asegurar también que los contenidos del resto de ficheros que formen un ensamblado no hayan sido alterados lo que se hace es calcular el código de dispersión de éstos antes de cifrar el ensamblado y guardarlo en el elemento correspondiente a cada fichero en la tabla FileDef del manifiesto. El algoritmo de cifrado usado por defecto es SHA-1, aunque en este caso también se da la posibilidad de usar MD5. En ambos casos, cada vez que se accede al fichero para acceder a un tipo o recurso se calculará de nuevo su valor de dispersión y se comprobará que coincida con el almacenado en FileDef.

Dado que las claves públicas son valores que ocupan muchos bytes (2048 bits), lo que se hace para evitar que los metadatos sean excesivamente grandes es no incluir en las referencias a ensamblados externos de la tabla AssemblyRef las claves públicas de dichos ensamblados, sino sólo los 64 últimos bits resultantes de aplicar un algoritmo de dispersión a dichas claves. A este valor recortado se le llama **marca de clave pública**.

Hay dos tipos de ensamblados: **ensamblados privados** y **ensamblados compartidos**. Los privados se almacenan en el mismo directorio que la aplicación que los usa y sólo puede usarlos ésta, mientras que los compartidos se almacenan en un **caché de ensamblado global** (GAC) y pueden usarlos cualquiera que haya sido compilada referenciándolos.

Los compartidos han de cifrarse con RSA ya que lo que los identifica es en el GAC es su nombre (sin extensión) más su clave pública, lo que permite que en el GAC puedan instalarse varios ensamblados con el mismo nombre y diferentes claves públicas. Es decir, es como si la clave pública formase parte del nombre del ensamblado, razón por la que a los ensamblados así cifrados se les llama **ensamblados de nombre fuerte**. Esta política permite resolver los conflictos derivados de que se intente instalar en un mismo equipo

varios ensamblados compartidos con el mismo nombre pero procedentes de distintas empresas, pues éstas tendrán distintas claves públicas.

También para evitar problemas, en el GAC se pueden mantener múltiples versiones de un mismo ensamblado. Así, si una aplicación fue compilada usando una cierta versión de un determinado ensamblado compartido, cuando se ejecute sólo podrá hacer uso de esa versión del ensamblado y no de alguna otra más moderna que se hubiese instalado en el GAC. De esta forma se soluciona el problema del **infierno de las DLL** comentado al principio del tema.

En realidad es posible modificar tanto las políticas de búsqueda de ensamblados (por ejemplo, para buscar ensamblados privados fuera del directorio de la aplicación) como la política de aceptación de ensamblados compartidos (por ejemplo, para que se haga automáticamente uso de las nuevas versiones que se instalen de DLLs compartidas) incluyendo en el directorio de instalación de la aplicación un fichero de configuración en formato XML con las nuevas reglas para las mismas. Este fichero ha de llamarse igual que el ejecutable de la aplicación pero ha de tener extensión **.cfg**.

1.5 Librería de clase base (BCL)

La Librería de Clase Base (BCL) es una librería incluida en el *.NET Framework* formada por cientos de tipos de datos que permiten acceder a los servicios ofrecidos por el CLR y a las funcionalidades más frecuentemente usadas a la hora de escribir programas. Además, a partir de estas clases prefabricadas el programador puede crear nuevas clases que mediante herencia extiendan su funcionalidad y se integren a la perfección con el resto de clases de la BCL. Por ejemplo, implementando ciertas interfaces podemos crear nuevos tipos de colecciones que serán tratadas exactamente igual que cualquiera de las colecciones incluidas en la BCL.

Esta librería está escrita en MSIL, por lo que puede usarse desde cualquier lenguaje cuyo compilador genere MSIL. A través de las clases suministradas en ella es posible desarrollar cualquier tipo de aplicación, desde las tradicionales aplicaciones de ventanas, consola o servicio de Windows NT hasta los novedosos servicios Web y páginas ASP.NET. Es tal la riqueza de servicios que ofrece que puede crearse lenguajes que carezcan de librería de clases propia y sólo usen la BCL -como C#.

1.6 Concepto de espacio de nombres

Dado la amplitud de la BCL, ha sido necesario organizar las clases en ella incluida en **espacios de nombres** que agrupen clases con funcionalidades similares.

Del mismo modo que los ficheros se organizan en directorios, los tipos de datos se organizan en **espacio de nombres**.

Por un lado estos espacios permiten tener más organizados los tipos de datos, lo que facilita su localización. De hecho, esta es la forma en que se encuentra organizada la BCL, de modo que todas las clases más comúnmente usadas en cualquier aplicación pertenecen al espacio de nombres llamado **System**, las de acceso a bases de datos en **System.Data**, las de realización de operaciones de entrada/salida en **System.IO**, etc.

Por otro lado, los espacios de nombres también permiten poder usar en un mismo programa varias clases con igual nombre si pertenecen a espacios diferentes. La idea es que cada fabricante defina sus tipos dentro de un espacio de nombres propio para que así no haya conflictos si varios fabricantes definen clases con el mismo nombre y se quieren usar a la vez en un mismo programa. Obviamente para que esto funcione no han de coincidir los nombres los espacios de cada fabricante, y una forma de conseguirlo es dándoles el nombre de la empresa fabricante, o su nombre de dominio en Internet, etc.

Por ejemplo, los espacios de nombres más usados son:

Espacio de nombres	Utilidad de los tipos de datos que contiene
System	Tipos muy frecuentemente usados, como los tipos básicos, tablas, excepciones, fechas, números aleatorios, recolector de basura, entrada/salida en consola, etc.
System.Collections	Colecciones de datos de uso común como pilas, colas, listas, diccionarios, etc.
System.Data	Manipulación de bases de datos. Forman la denominada arquitectura ADO.NET .
System.IO	Manipulación de ficheros y otros flujos de datos.
System.Net	Realización de comunicaciones en red.
System.Reflection	Acceso a los metadatos que acompañan a los módulos de código.
System.Runtime.Remoting	Acceso a objetos remotos.
System.Security	Acceso a la política de seguridad en que se basa el CLR.
System.Threading	Manipulación de hilos.
System.Web.UI.WebControls	Creación de interfases de usuario basadas en ventanas para aplicaciones Web.
System.Windows.Forms	Creación de interfases de usuario basadas en ventanas para aplicaciones estándar.
System.Xml	Acceso a datos en formato XML.

Tabla 1: Espacios de nombres de la BCL más usados

1.7 Common Type System (CTS)

El **Common Type System** (CTS) o Sistema de Tipo Común es el conjunto de reglas que han de seguir las definiciones de tipos de datos para que el CLR las acepte. Es decir, aunque cada lenguaje gestionado disponga de su propia sintaxis para definir tipos de datos, en el MSIL resultante de la compilación de sus códigos fuente se ha de cumplir las reglas del CTS. Algunos ejemplos de estas reglas son:

- Cada tipo de dato puede constar de cero o más miembros. Cada uno de estos miembros puede ser un campo, un método una propiedad o un evento.
- No puede haber herencia múltiple, y todo tipo de dato ha de heredar directa o indirectamente de **System.Object**.
- Los modificadores de acceso admitidos son:

Modificador	Código desde el que es accesible el miembro
public	Cualquier código
Private	Código del mismo tipo de dato
Family	Código del mismo tipo de dato o de hijos de éste.
Assembly	Código del mismo ensamblado
family and assembly	Código del mismo tipo o de hijos de éste ubicado en el mismo ensamblado
family or assembly	Código del mismo tipo o de hijos de éste, o código ubicado en el mismo ensamblado

Tabla 2: Modificadores de acceso a miembros admitidos por el CTS

1.8 Common Language Specification (CLS)

El **Common Language Specification** (CLS) o Especificación del Lenguaje Común es un conjunto de reglas que han de seguir las definiciones de tipos que se hagan usando un determinado lenguaje gestionado si se desea que sean accesibles desde cualquier otro lenguaje gestionado. Obviamente, sólo es necesario seguir estas reglas en las definiciones de tipos y miembros que sean accesibles externamente, y no la en las de los privados. Además, si no importa la interoperabilidad entre lenguajes tampoco es necesario seguirlas. A continuación se listan algunas de reglas significativas del CLS:

- Los tipos de datos básicos admitidos son **bool, char, byte, short, int, long, float, double, string** y **object**. Nótese pues que no todos los lenguajes tienen porqué admitir los tipos básicos enteros sin signo o el tipo **decimal** como lo hace C#.
- Las tablas han de tener una o más dimensiones, y el número de dimensiones de cada tabla ha de ser fijo. Además, han de indexarse empezando a contar desde 0.
- Se pueden definir tipos abstractos y tipos sellados. Los tipos sellados no pueden tener miembros abstractos.
- Las excepciones han de derivar de **System.Exception**, los delegados de **System.Delegate**, las enumeraciones de **System.Enum**, y los tipos por valor que no sean enumeraciones de **System.ValueType**.
- Los métodos de acceso a propiedades en que se traduzcan las definiciones get/set de éstas han de llamarse de la forma **get_X** y **set_X** respectivamente, donde X es el nombre de la propiedad; los de acceso a indizadores han de traducirse en métodos **get_Item** y **setItem**; y en el caso de los eventos, sus definiciones add/remove han de traducirse en métodos de **add_X** y **remove_X**.
- En las definiciones de atributos sólo pueden usarse enumeraciones o datos de los siguientes tipos: **System.Type, string, char, bool, byte, short, int, long, float, double** y **object**.
- En un mismo ámbito no se pueden definir varios identificadores cuyos nombres sólo difieran en la capitalización usada. De este modo se evitan problemas al acceder a ellos usando lenguajes no sensibles a mayúsculas.
- Las enumeraciones no pueden implementar interfases, y todos sus campos han de ser estáticos y del mismo tipo. El tipo de los campos de una enumeración sólo puede ser uno de estos cuatro tipos básicos: **byte, short, int** o **long**.

2 INTRODUCCIÓN A C#.

2.1 Origen y necesidad de un nuevo lenguaje

C# (leído en inglés "C Sharp" y en español "C Almohadilla") es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el **lenguaje nativo de .NET**

La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

Un lenguaje que hubiese sido ideal utilizar para estos menesteres es Java, pero debido a problemas con la empresa creadora del mismo -Sun-, Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado añadirle para mejorarlo aún más y hacerlo un lenguaje orientado al desarrollo de componentes.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el *.NET Framework SDK*

2.2 Características de C#

Con la idea de que los programadores más experimentados puedan obtener una visión general del lenguaje, a continuación se recoge de manera resumida las principales características de C#. Algunas de las características aquí señaladas no son exactamente propias del lenguaje sino de la plataforma .NET en general. Sin embargo, también se comentan aquí también en tanto que tienen repercusión directa en el lenguaje, aunque se indicará explícitamente cuáles son este tipo de características cada vez que se toquen:

- **Sencillez:** C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. Por ejemplo:
 - El código escrito en C# es **autocontenido**, lo que significa que no necesita de ficheros adicionales al propio fuente tales como ficheros de cabecera o ficheros IDL
 - El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.
 - No se incluyen elementos poco útiles de lenguajes como C++ tales como macros, herencia múltiple o la necesidad de un operador diferente del punto (.) acceder a miembros de espacios de nombres (::)

- **Modernidad:** C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose son muy útiles para el desarrollo de aplicaciones y que en otros lenguajes como Java o C++ hay que simular, como un tipo básico **decimal** que permita realizar operaciones de alta precisión con reales de 128 bits (muy útil en el mundo financiero), la inclusión de una instrucción **foreach** que permita recorrer colecciones con facilidad y es ampliable a tipos definidos por el usuario, la inclusión de un tipo básico **string** para representar cadenas o la distinción de un tipo **bool** específico para representar valores lógicos.
- **Orientación a objetos:** Como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos, aunque eso es más bien una característica del CTS que de C#. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

C# soporta todas las características propias del paradigma de programación orientada a objetos: **encapsulación, herencia y polimorfismo**.

En lo referente a la encapsulación es importante señalar que aparte de los típicos modificadores **public**, **private** y **protected**, C# añade un cuarto modificador llamado **internal**, que puede combinarse con **protected** e indica que al elemento a cuya definición precede sólo puede accederse desde su mismo ensamblado.

Respecto a la herencia -a diferencia de C++ y al igual que Java- C# sólo admite herencia simple de clases ya que la múltiple provoca más quebraderos de cabeza que facilidades y en la mayoría de los casos su utilidad puede ser simulada con facilidad mediante herencia múltiple de interfases. De todos modos, esto vuelve a ser más bien una característica propia del CTS que de C#.

Por otro lado y a diferencia de Java, en C# se ha optado por hacer que todos los métodos sean por defecto sellados y que los redefinibles hayan de marcarse con el modificador **virtual** (como en C++), lo que permite evitar errores derivados de redefiniciones accidentales. Además, un efecto secundario de esto es que las llamadas a los métodos serán más eficientes por defecto al no tenerse que buscar en la tabla de funciones virtuales la implementación de los mismos a la que se ha de llamar. Otro efecto secundario es que permite que las llamadas a los métodos virtuales se puedan hacer más eficientemente al contribuir a que el tamaño de dicha tabla se reduzca.

- **Orientación a componentes:** La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. Es decir, la sintaxis de C# permite definir cómodamente **propiedades** (similares a campos de acceso controlado), **eventos** (asociación controlada de funciones de respuesta a notificaciones) o **atributos** (información sobre un tipo o sus miembros)
- **Gestión automática de memoria:** Todo lenguaje de .NET tiene a su disposición el recolector de basura del CLR. Esto tiene el efecto en el lenguaje de que no es necesario incluir instrucciones de destrucción de objetos. Sin embargo, dado que la destrucción de los objetos a través del recolector de basura es indeterminista y sólo se realiza cuando éste se active -ya sea por falta de memoria, finalización de la aplicación o solicitud explícita en el fuente-, C# también proporciona un mecanismo de liberación de recursos determinista a través de la instrucción **using**.
- **Seguridad de tipos:** C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que permite evita que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto y es especialmente necesario en un entorno gestionado por un recolector de basura. Para ello se toman medidas del tipo:

- Sólo se admiten **conversiones entre tipos compatibles**. Esto es, entre un tipo y antecesores suyos, entre tipos para los que explícitamente se haya definido un operador de conversión, y entre un tipo y un tipo hijo suyo del que un objeto del primero almacenase una referencia del segundo (**downcasting**) Obviamente, lo último sólo puede comprobarlo en tiempo de ejecución el CLR y no el compilador, por lo que en realidad el CLR y el compilador colaboran para asegurar la corrección de las conversiones.
- No se pueden usar **variables no inicializadas**. El compilador da a los campos un valor por defecto consistente en ponerlos a cero y controla mediante análisis del flujo de control del fuente que no se lea ninguna variable local sin que se le haya asignado previamente algún valor.
- Se comprueba que todo **acceso a los elementos de una tabla** se realice con índices que se encuentren dentro del rango de la misma.
- Se puede controlar la **producción de desbordamientos** en operaciones aritméticas, informándose de ello con una excepción cuando ocurra. Sin embargo, para conseguirse un mayor rendimiento en la aritmética estas comprobaciones no se hacen por defecto al operar con variables sino sólo con constantes (se pueden detectar en tiempo de compilación)
- A diferencia de Java, C# incluye **delegados**, que son similares a los punteros a funciones de C++ pero siguen un enfoque orientado a objetos, pueden almacenar referencias a varios métodos simultáneamente, y se comprueba que los métodos a los que apunten tengan parámetros y valor de retorno del tipo indicado al definirlos.
- Pueden definirse métodos que admitan un número indefinido de parámetros de un cierto tipo, y a diferencia lenguajes como C/C++, en C# siempre se comprueba que los valores que se les pasen en cada llamada sean de los tipos apropiados.
- **Instrucciones seguras:** Para evitar errores muy comunes, en C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes. Por ejemplo, la guarda de toda condición ha de ser una expresión condicional y no aritmética, con lo que se evitan errores por confusión del operador de igualdad (==) con el de asignación (=); y todo caso de un **switch** ha de terminar en un **break** o **goto** que indique cuál es la siguiente acción a realizar, lo que evita la ejecución accidental de casos y facilita su reordenación.
- **Sistema de tipos unificado:** A diferencia de C++, en C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada **System.Object**, por lo que dispondrán de todos los miembros definidos en ésta clase (es decir, serán "objetos")

A diferencia de Java, en C# esto también es aplicable a los tipos de datos básicos. Además, para conseguir que ello no tenga una repercusión negativa en su nivel de rendimiento, se ha incluido un mecanismo transparente de **boxing** y **unboxing** con el que se consigue que sólo sean tratados como objetos cuando la situación lo requiera, y mientras tanto puede aplicárseles optimizaciones específicas.

El hecho de que todos los tipos del lenguaje deriven de una clase común facilita enormemente el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.

- **Extensibilidad de tipos básicos:** C# permite definir, a través de **estructuras**, tipos de datos para los que se apliquen las mismas optimizaciones que para los tipos de datos básicos. Es decir, que se puedan almacenar directamente en pila (luego su creación, destrucción y acceso serán más rápidos) y se asignen por valor y no por referencia. Para conseguir que lo último no tenga efectos negativos al pasar estructuras como parámetros de métodos, se da la posibilidad de pasar referencias a pila a través del modificador de parámetro **ref**.

- **Extensibilidad de operadores:** Para facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través de las estructuras estén al mismo nivel que los básicos predefinidos en el lenguaje, al igual que C++ y a diferencia de Java, C# permite redefinir el significado de la mayoría de los operadores -incluidos los de conversión, tanto para conversiones implícitas como explícitas- cuando se apliquen a diferentes tipos de objetos.

Las redefiniciones de operadores se hacen de manera inteligente, de modo que a partir de una única definición de los operadores ++ y -- el compilador puede deducir automáticamente como ejecutarlos de manera prefija y postfixa; y definiendo operadores simples (como +), el compilador deduce cómo aplicar su versión de asignación compuesta (+=) Además, para asegurar la consistencia, el compilador vigila que los operadores con opuesto siempre se redefinan por parejas (por ejemplo, si se redefine ==, también hay que redefinir !=)

También se da la posibilidad, a través del concepto de **indizador**, de redefinir el significado del operador [] para los tipos de dato definidos por el usuario, con lo que se consigue que se pueda acceder al mismo como si fuese una tabla. Esto es muy útil para trabajar con tipos que actúen como colecciones de objetos.

- **Extensibilidad de modificadores:** C# ofrece, a través del concepto de **atributos**, la posibilidad de añadir a los metadatos del módulo resultante de la compilación de cualquier fuente información adicional a la generada por el compilador que luego podrá ser consultada en tiempo ejecución a través de la librería de reflexión de .NET . Esto, que más bien es una característica propia de la plataforma .NET y no de C#, puede usarse como un mecanismo para definir nuevos modificadores.
- **Versionable:** C# incluye una **política de versionado** que permite crear nuevas versiones de tipos sin temor a que la introducción de nuevos miembros provoquen errores difíciles de detectar en tipos hijos previamente desarrollados y ya extendidos con miembros de igual nombre a los recién introducidos.

Si una clase introduce un nuevo método cuyas redefiniciones deban seguir la regla de llamar a la versión de su padre en algún punto de su código, difícilmente seguirían esta regla miembros de su misma signatura definidos en clases hijas previamente a la definición del mismo en la clase padre; o si introduce un nuevo campo con el mismo nombre que algún método de una clase hija, la clase hija dejará de funcionar. Para evitar que esto ocurra, en C# se toman dos medidas:

- Se obliga a que toda redefinición deba incluir el modificador **override**, con lo que la versión de la clase hija nunca sería considerada como una redefinición de la versión de miembro en la clase padre ya que no incluiría **override**. Para evitar que por accidente un programador incluya este modificador, sólo se permite incluirlo en miembros que tengan la misma signatura que miembros marcados como redefinibles mediante el modificador **virtual**. Así además se evita el error tan frecuente en Java de creerse haber redefinido un miembro, pues si el miembro con **override** no existe en la clase padre se producirá un error de compilación.
 - Si no se considera redefinición, entonces se considera que lo que se desea es ocultar el método de la clase padre, de modo que para la clase hija sea como si nunca hubiese existido. El compilador avisará de esta decisión a través de un mensaje de aviso que puede suprimirse incluyendo el modificador **new** en la definición del miembro en la clase hija para así indicarle explícitamente la intención de ocultación.
- **Eficiente:** En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y no permite el uso de punteros. Sin embargo, y a diferencia de Java, en C# es posible saltarse dichas restricciones manipulando objetos a través de punteros. Para ello basta marcar

regiones de código como inseguras (modificador **unsafe**) y podrán usarse en ellas punteros de forma similar a cómo se hace en C++, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad procesamiento muy grandes.

- **Compatible:** Para facilitar la migración de programadores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes, sino que el CLR también ofrece, a través de los llamados **Platform Invocation Services (PInvoke)**, la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos tales como las DLLs de la API Win32. Nótese que la capacidad de usar punteros en código inseguro permite que se pueda acceder con facilidad a este tipo de funciones, ya que éstas muchas veces esperan recibir o devuelven punteros.

También es posible acceder desde código escrito en C# a objetos COM. Para facilitar esto, el *.NET Framework SDK* incluye una herramienta llamada **tlbimp** y **regasm** mediante las que es posible generar automáticamente clases proxy que permitan, respectivamente, usar objetos COM desde .NET como si de objetos .NET se tratase y registrar objetos .NET para su uso desde COM.

Finalmente, también se da la posibilidad de usar controles ActiveX desde código .NET y viceversa. Para lo primero se utiliza la utilidad **aximp**, mientras que para lo segundo se usa la ya mencionada **regasm**.

3 INTRODUCCIÓN A XML.

Dado que para la especificación de los flujos, hemos decidido apegarnos al estándar XPD (XML Process Definition Language), de la WFM (Workflow Management Coalition), tendremos cada uno de los flujos a administrar por nuestro motor expresado en tal lenguaje.

Debemos considerar que en esta primera etapa del proyecto, se desea definir que subconjunto del XPD, soportará nuestro Workflow (dado que en general, no se soporta el estándar completo como tal), para generar una primera versión que brinde las funcionalidades más utilizadas que debe ofrecer todo motor de WF, tenemos que definir la forma en que administraremos nuestras definiciones de procesos, como parte de la evaluación del alcance planteado.

A la hora de enfrentar la administración y parseo de documentos XML, tenemos diferentes propuestas, diferentes encares posibles, o formas de parsearlo. En tal sentido debemos tomar una serie de decisiones y, el presente documento intenta justamente, presentar las alternativas que brinda .NET y hacer un análisis de fortalezas y debilidades de cada una de ellas. Una vez presentadas las técnicas y sus principales características, deberíamos evaluar cual seleccionar, no debiendo dejarse de lado, la posibilidad de adoptar una estrategia mixta.

Básicamente las posibilidades pasan por cargar el documento entero (Document Object Model) en memoria, para luego parsearlo de una forma cómoda, rápida y flexible o consumir el documento en un formato "ForwardOnly", bajo demanda, procesando el documento a medida que se consume, teniendo en cada momento, una pequeña fracción del mismo cargado en memoria.

Las opciones serían:

- Cargar todo el DOM en memoria.
- Cargar el documento a medida que lo necesitamos. Si decidimos hacer esto tenemos dos opciones.
 - Modelo Push. EL modelo que provee SAX.
 - Modelo Pull. El modelo que provee .NET a través de XMLStreamReader.

Una tercera alternativa sería utilizar parsers ya desarrollados en MSXML, vía Interop. Si tuviéramos una gran artillería de código ya desarrollada, destinada a parsear documentos de este estilo en MSXML - COM, podríamos usar la puerta abierta, que en .NET tenemos (Interop) para llamar aplicaciones COM y, precisamente utilizar esta artillería, para parsear el documento. Dado que no es el caso, parece claro que no es conveniente desarrollar un parseo utilizando MSXML3 y luego llamarlo utilizando Interop.

3.1 En memoria vs. ForwardOnly

Claramente, la principal variable a considerar a la hora de tomar esta decisión, pasa por el consumo de memoria y, por ende, por el tamaño de los documentos XML que vamos a manejar. Por lo que se ha concluido en el estudio de XPD, se generan documentos bastante extensos pero sin llegar a un tamaño tal, que invalide la técnica descargar el documento en memoria.

La técnica de cargar todo un modelo de objetos (DOM) que represente al documento XML y su contenido en memoria, permite mucha mayor flexibilidad a la hora de administrarlo. Una de las técnicas más utilizadas en el parseo de documentos XML, es la utilización de XPath. XPath permite seleccionar sub-secciones del documento, no necesariamente contiguas, que cumplen con determinadas características que son relevantes para nosotros en un momento dado. XPath, solo se aplica sobre un DOM en memoria y no es válido en una técnica ForwardOnly.

En el polo opuesto tenemos a la técnica que supone procesar el documento a medida que se lee, cargando solo pequeñas fracciones del mismo en memoria, con un consume sensiblemente menor. Otra ventaja que brinda este segundo modelo, pasa por la posibilidad de comenzar a procesar el documento a medida que se lee, sin necesidad de esperar a cargarlo todo, para comenzar a procesarlo.

Veremos ahora, cada una de las técnicas en detalle para brindar argumentos que ayuden a la decisión, a la vez que quede como marco de referencia para el momento del desarrollo.

3.2 Cargando el documento en un formato ForwardOnly

Modelo Push vs. el Modelo Pull de .NET:

Si decidimos trabajar en un formato ForwardOnly, tenemos dos alternativas a las que comúnmente se denomina técnica de *push* y técnica de *pull*.

PUSH: La forma en que se manejaba un documento XML anteriormente a .NET, era mediante SAX(Simple API for XML). SAX es un modelo *basado en eventos* que realiza el “pushing out” de información de determinados nodos que busca en un documento. La información que presenta es recibida por un “content handler”. Al content handler es a quien debemos dar toda la capacidad lógica de capturar los eventos que genera el proceso de lectura del XML, y administrar el contenido del documento basado en estos eventos.

PULL: La contrapartida a esta propuesta es utilizar una técnica de pull. Si bien .NET, indirectamente, brinda el soporte necesario para que podamos desarrollar nosotros la técnica de push, presenta una técnica que parece más adecuada, la técnica de pull. La idea es recorrer el documento a medida que sea necesario, teniendo mayor control del flujo sobre el documento que en el modelo anterior de eventos.

El libro plantea una serie de puntos donde la técnica de pull, que nos brinda .NET aventaja a la técnica de push, que enumeramos a continuación:

- **Manejo de estado:** En el modelo push, el content handler (el que recibe la información que los lectores brindan) debe construir complejas máquinas de estado, mientras que el cliente del modelo pull, maneja el estado con un refinamiento procedural top-down mucho más natural.
- **Múltiples Entradas:** El modelo pull, facilita mucho más la lectura de varios streams de entrada simultáneos.
- Es muy sencillo **construir** un modelo push, **sobre** un pull. No lo es a la inversa.
- **Elementos importantes:** El modelo de push, libera al content handler, “todo lo que encuentra”, por decirlo de alguna forma. El modelo pull, permite saltarse algunos elementos, cuando no son necesarios. Por ejemplo, podríamos no leer los atributos, si no nos interesa.
- El modelo pull, permite ir procesando cosas **a medida que son cargadas**, sin esperar a cargar todo el documento.
- Podemos obviar buena parte del trabajo, e **ir directamente** a determinada parte del documento que nos interese, si sabemos como llegar a ella, navegando a través del XML, utilizando el DOM o rutas XPath, que analizaremos luego.

Parece estar bastante claro que, si decidimos utilizar la técnica de cargado del documento bajo demanda, la técnica de pull es la más adecuada. En tal sentido deberíamos utilizar la clase XmlTextReader del assembly System.XML

XmlTextReader

Para crear un reader existen varias opciones, pero la más sencilla parece ser a partir del Path al XML.

```
XmlTextReader reader = new XmlTextReader(path); // Donde path puede ser una dirección en disco, por ejemplo.
```

Atributos:

Presenta una serie de atributos, no todos útiles en todas las circunstancias, que representarían lo que nosotros quisiéramos saber, para cada nodo del XML.

- AttributeCount: Cantidad de atributos que tiene el nodo.
- Base URI: Retorna la URI base del nodo.
- Depth: Devuelve el camino al nodo actual a través del stack de XML.
- EOF: Retorna verdadero si llegamos al final del archivo.
- HasAttributes: Retorna verdadero si el nodo actual tiene atributos y falso en caso contrario.
- HasValue: Retorna true, si el nodo actual tiene valor y falso en caso contrario.
- IsEmptyElement: Retorna verdadero si el elemento es vacío.
- LineNumber: Número de línea del cursor, dentro del archivo XML.
- LinePosition: Posición del cursor, en la línea del archivo.
- Name: Nombre del nodo actual.
- NodeType: Tipo de nodo. Los tipos de nodo más útiles son Element, EndElement, Text, Entity, EndEntity, Document, Comment, WhiteSpace, etc.
- Value: Valor del nodo actual, en formato string.

Si bien hay más atributos, parecen ser estos, los más útiles para nuestro objetivo.

Miembros:

- Close: Termina la lectura, cierra la conexión con el "source" (sea base de datos, archivo, etc.).
- IsStartElement: Retorna verdadero, si el nodo actual es el comienzo de un elemento.
- MoveToAttribute: Se posiciona en el atributo del nodo actual, identificado por su índice o nombre.
- MoveToContent: Avanza el puntero en el XML, hasta el primer nodo cuyo tipo pertenezca al siguiente subconjunto de los tipos (text, CDATA, Element, EndElement, EntityReference, EndEntity).
- MoveToElement: Después de navegar por los atributos de un nodo, se puede volver al nodo base.
- MoveToNextAttribute: Si existe un siguiente elemento, avanza el puntero al mismo y devuelve true. Si no existen más atributos, devuelve falso.
- Read: Si estás apuntando al fin de archivo devuelve false. En caso contrario, lee n nodo (queda apuntando a él) y devuelve true. Normalmente se usa como condición booleana de un while.
- ReadAttributeValue: Lee y parsea el valor de un atributo en un elemento de tipo "Text" o "EntityReference".
- ReadElementString: Devuelve todo el elemento en un string. Solo aplicable a nodos de tipo "Text".
- ReadEndElement: Se consume un nodo de tipo "EndElement", avanzando al cabecal del siguiente elemento. Devuelve true, si estamos ante un "EndElement", devuelve falso en caso contrario.
- ReadInnerXml: Lee todo el contenido, incluyendo las marcas, del nodo actual como un string. Si el nodo actual es vacío, devuelve el string vacío.
- ReadOuterXml: Lee todo el contenido, incluyendo las marcas, del nodo actual como un string. Si el nodo actual es vacío, devuelve el string vacío. A diferencia del "Inner" devuelve también las marcas de principio y fin.
- ReadStartElement: Verifica que el nodo actual sea un "Element" y posiciona el puntero en el siguiente nodo.
- ResolveEntity: Resuelve la referencia a una entidad para nodos de tipo "EntityReference".
- Skip: Avanza al siguiente nodo ignorando el actual.

3.3 DOM

Más System.XML assembly:

.NET provee un assembly, System.XML que contiene las principales clases de acceso a XML. La siguiente es una lista de las mismas:

- XmlAttribute: Representa los nodos de tipo atributo de un documento XML.
- XmlDocument: Es utilizada para acceder al documento XML e identificar el elemento root. Es el puente de acceso al documento si utilizamos la representación de DOM en memoria.
- XmlElement: Representa los nodos de tipo elemento en un documento XML.
- XmlNode: Es el objeto principal utilizado en el DOM. Es una clase abstracta, por lo que no puede ser instanciada. Si define varias cosas útiles, como los atributos, el nombre, etc.
- XmlNamedNodeMap: Es una colección de valores asociados a atributos.
- XmlNodeList: Representa una lista de nodos de un XML.
- XmlNodeReader: Clase específica para leer un elemento de un documento XML.
- XmlTextWriter: Permite escribir un documento XML, en un archivo, de forma secuencial.
- XmlValidatingReader: Permite validar, de forma sumamente sencilla, un documento XML, contra un determinado formato específico que éste debe cumplir. El formato, como veremos luego, puede ser expuesto de distintas formas.
- XslTransform: Transformación XSL, aplicar a un XML.

Veamos algunas de ellas, las que parecen ser más útiles para nuestro problema.

XMLNodeTypeEnumeration

Enumeración de los tipos de nodo que podemos encontrar en la administración de un DOM. Toma los siguientes valores:

Attribute	Comment	Document
Element	EndElement	Entity
EndElement	Text	WhiteSpace
SignificantWhiteSpace	EntityReference	EndReference

XmlNode

Clase abstracta que representa un nodo de un documento XML. Si es un nodo de tipo Attribute, por decir algo, es un elemento de la clase XmlAttribute. Si es de tipo Comment, se puede asignar a un objeto de la clase XmlComment. Si es un Element, lo asignaremos a un objeto de la clase XmlElement, y así con XmlEntity, XmlEntityReference, XmlProcessingInstruction, XmlText, etc. Cada una de estas clases hereda de XmlNode, por lo que redefine de forma adecuada, cada una de sus propiedades y atributos.

XMLDocument

Para cargar un documento XML en memoria (en una estructura de DOM), debe utilizarse un objeto de esta clase. Provee la posibilidad de cargar el documento e identificar su nodo root. Hereda de XmlNode. Agrega a los métodos de esta clase, los métodos Load y LoadXml, que permiten generar el DOM a partir de un string con el XML, o de un documento en algún dispositivo de almacenamiento.

Atributos:

Attributes	Retorna una XmlAttributeCollection, conteniendo la lista de atributos de un determinado nodo. Aplica solo a nodos de tipo element.
BaseURI	Retorna la URI base del nodo actual.
ChildNodes	Devuelve todos los hijos del nodo actual.
DocumentElement	Retorna el elemento root (de la clase XmlElement) o el documento XML propiamente dicho.
FirstChild	Devuelve el primer nodo hijo del nodo actual.
HasChildNodes	Indica si el nodo actual tiene hijos.
Implementation	Retorna el objeto XmlImplementation del documento XML.
Item	Devuelve el primer elemento hijo identificado por un determinado nombre o por un NamespaceURI.
LastChild	Retorna el último hijo del nodo actual.
LocalName	Retorna el nombre del nodo actual con el prefijo del namespace.
Name	Retorna el nombre del nodo actual.
NamespaceURI	Retorna el namespaceURI del nodo actual.
NodeType	Retorna el tipo del nodo actual.
OwnerDocument	Retorna el XmlDocument que contiene el nodo actual.
ParentNode	Retorna el nodo padre del nodo actual. Los tipos de nodos que no tienen padres (como por ejemplo los atributos, retornan NULL).
Prefix	Obtiene o setea el prefijo del namespace del nodo actual.
Value	Obtiene o setea el valor del nodo actual.

Métodos:

Los métodos más útiles parecen ser los siguientes:

AppendChild (newNode)	Agrega el nodo como hijo del nodo actual.
CreateAttribute(name)	Agrega un atributo al nodo actual, con el nombre especificado por parámetro. De forma similar tenemos los métodos CreateComment, CreateElement, CreateDocumentFragment, CreateEntityReference, etc.
CreateNavigator()	Crea un XpathNavigator, utilizado para navegar el documento.
GetElementsByTagName(name)	Devuelve la lista de los elementos que machean con el nombre.
Load(filePath o URL)	Carga un DOM a partir de una referencia a un archivo XML
RemoveAll	Elimina todos los hijos y/o atributos del nodo actual.
SelectNodes(xpath)	Retorna una colección de elementos que machean con el xpath.

3.4 XPath

XPath, es un mecanismo para direccionar partes de un XML, de forma ágil y rápida. El "path", es por su similitud con la noción de path de un sistema de archivos. En el file system, por ejemplo, \Book\Chap13 identifica el subdirectorio correspondiente al capítulo 13 del directorio root "Book". En un documento XML, /Guitars/Guitar identifica a todos los elementos llamados Guitar, hijos del elemento root Guitars. Las expresiones de XPath son completamente descritas en la especificación XPath que se encuentra en: <http://www.w3.org/TR/xpath>.

La idea general es trabajar con expresiones que funcionan como filtros, que nos devuelven solo determinadas partes del documento XML.

Una de las utilidades de XPath, es la extracción de información de documentos XML. La FCL viene con un "motor" de XPath llamado System.Xml.XPath.XPathNavigator, pero antes de analizarlo, establezcamos conceptos generales de XPath.

Cosas básicas, para entender el funcionamiento de XPath:

Las expresiones son los bloques constructivos de XPath. El tipo más común de expresiones, son los path de ubicación (location path, en inglés). Un ejemplo de location path es **/Guitars/Guitar**, como ya vimos antes.

Para acceder a atributos, utilizamos el carácter @. Por ejemplo, accedemos todos los atributos llamados "Image", de nodos "Guitar", hijos de nodos "Guitars", de la siguiente forma **/Guitars/Guitar/@Image**

Podemos obtener todos los elementos Guitar, independientemente de sus padres:

```
//Guitar
```

La siguiente expresión, nos devolvería todos los nodos hijos de un nodo root, llamado "Guitars":

```
/Guitars/*
```

Con una combinación de dos casos anteriores podemos seleccionar todos los atributos de nodos "Guitar", en cualquier lugar del documento:

```
//Guitar/@*
```

Los componentes de un location path, se llaman location steps. Un location step consta de tres partes: un eje, un nodo test, y cero o más predicados. La forma general para un location step es:

```
eje::node-test[predicado1][predicado2][...]
```

El eje describe una relación existente entre los nodos. Algunos ejes posibles son **child**, **descendant**, **descendant-or-self**, **parent**, **ancestor**, y **ancestor-or-self**. El eje por defecto es **child**. Más aún, la expresión **/Guitars/Guitar** puede también ser escrita como:

```
/child::Guitars/child::Guitar
```

La siguiente expresión evalúa a todos los nodos "Guitar" que son descendientes del root o son el propiamente un root del documento: **/descendant-or-self::Guitar**

La expresión ***//Guitar/@**** puede ser escrita como ***//Guitar/attribute::****

Los predicados, cuando aparecen, lo hacen entre paréntesis rectos. Por ejemplo, la siguiente expresión evalúa a todos los elementos "Guitar" en el documento, que tienen un atributo "Image" cuyo valor es "MyStrat.jpeg":

```
//Guitar[@Image = "MyStrat.jpeg"]
```

Los predicados pueden incluir los siguientes operadores de comparación: <, >, =, !=, <=, and >=. Por ejemplo, ***//Guitar[Year > 1980][Make = "Fender"]***, evalúa a lo mismo que ***//Guitar[Year > 1980 and Make = "Fender"]***.

XPath soporta otro conjunto de funciones intrínsecas que no siempre son soportadas por los predicados. La forma de obtener todos los elementos "Guitar", que tienen atributos make, que comiencen con G, sería: ***//Guitar[starts-with (Make, "G")]***

Cuando se ejecuta una sentencia XPath, que dado un location path, devuelve un conjunto de nodos, XPath como DOM, utiliza una estructura de árbol, para almacenar los nodos.

Nuevamente, si ejecutamos ***//Guitar***, aplicado al XML:

```
<?xml version="1.0"?>
<Guitars>
  <Guitar Image="MySG.jpeg">
    <Make>Gibson</Make>
    <Model>SG</Model>
    <Year>1977</Year>
    <Color>Tobacco Sunburst</Color>
    <Neck>Rosewood</Neck>
  </Guitar>
  <Guitar Image="MyStrat.jpeg" PreviousOwner="Eric Clapton">
    <Make>Fender</Make>
    <Model>Stratocaster</Model>
    <Year>1990</Year>
    <Color>Black</Color>
    <Neck>Maple</Neck>
  </Guitar>
</Guitars>
```

tenemos por resultado un set de dos nodos, cada uno de los cuales representa un nodo "Guitar". Cada uno es root de un nodo conteniendo Make, Model, Year, Color, and Neck. Cada uno de ellos tiene un nodo "text".

Los tipos de nodo de XPath son definidos por separado de los tipos de nodo de DOM, si bien comparten similitudes. XPath define menos tipos de nodo que DOM, lo que hace que los tipos de

nodo de XPath sean un subconjunto funcional de los tipos de nodo de DOM.

XpathNavigator y su funcionamiento:

El .NET Framework tiene el namespace System.Xml.XPath, que contiene clases para administrar XPath. Las principales son XPathDocument, que representa el documento XML que queremos consultar con XPath, XPathNavigator, la cual provee un mecanismo automático para evaluar consultas XPath, y XPathNodeIterator, que representa el conjunto de nodos resultantes y permite iterar sobre él.

Veamos un ejemplo, XPathDemo.cs, que leería el documento presentado antes

```
using System;
using System.Xml.XPath;
class MyApp
{
    static void Main ()
    {
        XPathDocument doc = new XPathDocument ("Guitars.xml");
        XPathNavigator nav = doc.CreateNavigator ();
        XPathNodeIterator iterator = nav.Select ("/Guitars/Guitar");
        while (iterator.MoveNext ()) {
            XPathNodeIterator it = iterator.Current.Select ("Make");
            it.MoveNext ();
            string make = it.Current.Value;
            it = iterator.Current.Select ("Model");
            it.MoveNext ();
            string model = it.Current.Value;
            Console.WriteLine ("{0} {1}", make, model);
        }
    }
}
```

3.5 ¿Cómo validar un documento XML?

Para validar un documento XML, parece ser lo más útil, utilizar la clase `XmlValidatingReader`. Un objeto de esta clase, se crea a partir del reader y lo que hace es, en pocas palabras, redefinir la función `Read` del `XmlTextReader`, para verificar, a medida que lee, que el XML cumple con el esquema definido. El esquema puede estar definido en un DTD, Esquema, XDR, etc. En nuestro caso, tenemos el esquema XSD.

Crear el validador:

Hay tres constructores definidos por `XmlValidatingReader`. Uno acepta un `XmlTextReader` como parámetro y validará lo que el reader lea. Los otros dos, mediante una serie de parámetros permiten manejar fragmentos de documentos, obtenidos como resultados de consultas a un SQL Server, por ejemplo.

```
XmlValidatingReader valid = new XmlValidatingReader(reader);
```

Definir el tipo de validación:

Existen varios tipos de validación de un documento XML, uno de los cuales es validar contra un esquema de definición (XSD), que es el que utilizaremos nosotros. Para setear el tipo de validación a un esquema hacemos:

```
Valid.ValidationType = ValidationType.Schema;
```

Definir los esquemas contra quienes validar:

El validador tiene una colección de esquemas contra los que valida. Debemos crear una colección con los esquemas que nos interesen (casi seguramente solo el esquema de XPDL) y asignarla a esta property del validador.

```
XmlSchemaCollection sCol = new XmlSchemaCollection();  
sCol.Add("http://www.wfmc.org/2002/XPDL1.0",  
        "D:/XMLRepository/ICWorkflow_XPDL.xml");  
valid.Schemas.Add(sCol);
```

Validar un Flujo, contra el esquema de XPDL:

El siguiente sería el código de una aplicación de consola, que valida un documento XML, contra un esquema XSD, precisamente lo que queremos hacer nosotros.

La llamada al ejemplo sería así: validate guitars.xml guitars.xsd

- Ejemplo de validación - Validate.cs -

```
using System;
using System.Xml;
using System.Xml.Schema;

class MyApp
{
    static void Main (string[] args)
    {
        if (args.Length < 2) {
            Console.WriteLine ("Syntax: VALIDATE xmldoc schemadoc");
            return;
        }

        XmlValidatingReader reader = null;

        try {
            XmlTextReader nvr = new XmlTextReader (args[0]);
            nvr.WhitespaceHandling = WhitespaceHandling.None;

            reader = new XmlValidatingReader (nvr);
            reader.Schemas.Add (GetTargetNamespace (args[1]), args[1]);
            reader.ValidationEventHandler +=
                new ValidationEventHandler (OnValidationError);

            while (reader.Read ());
        }
        catch (Exception ex) {
            Console.WriteLine (ex.Message);
        }
        finally {
            if (reader != null)
                reader.Close ();
        }
    }

    static void OnValidationError (object sender, ValidationEventArgs e)
    {
        Console.WriteLine (e.Message);
    }
}
```

```
public static string GetTargetNamespace (string src)
{
    XmlTextReader reader = null;

    try {
        reader = new XmlTextReader (src);
        reader.WhitespaceHandling = WhitespaceHandling.None;

        while (reader.Read ()) {
            if (reader.NodeType == XmlNodeType.Element &&
                reader.LocalName == "schema") {
                while (reader.MoveToNextAttribute ()) {
                    if (reader.Name == "targetNamespace")
                        return reader.Value;
                }
            }
        }
        return "";
    }
    finally {
        if (reader != null)
            reader.Close ();
    }
}
```

4 INTRODUCCIÓN ADO.NET

4.1 ADO.NET en la plataforma .NET Framework:

La mayoría de las aplicaciones necesitan algún mecanismo de acceso a datos. A la hora de crear una aplicación desde cero, se dispone de tres opciones excelentes para obtener acceso a los datos: ADO.NET, ADO y OLE DB. Más aún, en caso de tener una aplicación existente, pero que su ciclo esperado de vida sea de largo alcance, se debería considerar la posibilidad de rediseñar la tecnología de acceso a datos de la aplicación y utilizar ADO.NET en aplicaciones administradas o ADO en aplicaciones nativas. A largo plazo, el uso de las tecnologías más modernas de acceso a datos reduce el tiempo de desarrollo, simplifica el código y proporciona un rendimiento excelente.

Microsoft ha introducido recientemente una nueva plataforma para el desarrollo de aplicaciones —el .NET Framework. Microsoft presenta integrados en el concepto de Framework de desarrollo .NET el concepto de “the Common Language Runtime (CLR)”, lenguaje intermedio al que se pre-compilan todas las aplicaciones (independientemente del lenguaje de desarrollo original J++, C++, VB, C#, etc.), lo que permite desarrollar sobre una arquitectura multiplataforma (al estilo de Java), el concepto de “Active Server Pages+ (ASP+)”, el de formularios web y formularios Win, y una gran batería de librerías que componen lo que ellos llaman “the Class Framework”, o entorno de clases.

El Microsoft® .NET Framework (que vemos en la **Figura 1**) ofrece a los desarrolladores algunas ventajas sobre los ambientes COM tradicionales, incluyendo el manejo automático del tiempo de vida de los objetos (al estilo Garbage Colector de Java), librerías desarrolladas sobre el lenguaje intermedio, que permite utilizarlas de forma muy similar independientemente del lenguaje en el que se desarrolla, herencia cruzada entre lenguajes, manejo natural y unificado de excepciones, y la posibilidad de debuguear las aplicaciones cuando es necesario.

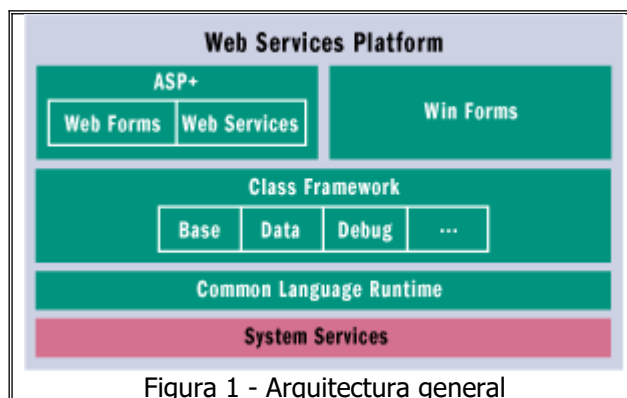


Figura 1 - Arquitectura general

En el diagrama anterior de la arquitectura de .NET, ADO.NET es la solución al módulo de “Data” que encontramos dentro de las clases del framework.

4.2 ADO.NET

ADO.NET constituye la interfaz fundamental de las aplicaciones para proporcionar servicios de acceso a datos en la plataforma Microsoft .NET. Puede utilizar ADO.NET para tener acceso a orígenes de datos mediante los nuevos proveedores de datos .NET, así como a proveedores de datos OLE DB existentes mediante el proveedor de datos OLE DB .NET. La interfaz ADO.NET está específicamente diseñada para aplicaciones Web basadas en mensajes, no obstante, proporciona también funcionalidad aplicable a otras arquitecturas de aplicaciones. ADO.NET admite el acceso a datos de correspondencia imprecisa y así permite maximizar el uso compartido de datos mediante la reducción del número de conexiones activas con la base de datos; de este modo, disminuye la probabilidad de que varios usuarios compitan entre sí por los recursos limitados del servidor de base de datos.

ADO.NET proporciona distintos tipos de acceso a datos. Si la aplicación Web o el servicio Web de XML necesita tener acceso a datos de múltiples orígenes, o interoperar con otras aplicaciones (tanto locales como remotas), o puede beneficiarse de la persistencia y transferencia de resultados almacenados en memoria caché, el conjunto de datos (objeto DataSet) es una opción excelente. Como alternativa, ADO.NET proporciona comandos de datos y lectores de datos para comunicarse directamente con el origen de datos. Las operaciones directas con la base de datos mediante comandos de datos (SqlCommand y OleDbCommand) y lectores de datos (DataReader) incluyen la ejecución de consultas y de procedimientos almacenados (stored procedures), la creación de objetos de base de datos y la actualización y eliminación directa utilizando comandos DDL.

Asimismo, ADO.NET maximiza el uso compartido de datos ya que admite formato de transmisión y persistencia basado en XML para el objeto fundamental de las aplicaciones ADO.NET distribuidas: el conjunto de datos. Un conjunto de datos es una estructura de datos relacionales de la que se puede leer, en la que se puede escribir, en la que se puede eliminar. Al mismo tiempo, se puede serializar un conjunto de datos, utilizando XML. Los conjuntos de datos de ADO.NET facilitan la creación de aplicaciones que requieran un intercambio de datos de correspondencia imprecisa entre niveles de aplicaciones y varios sitios Web, por ejemplo.

Como los conjuntos de datos son remotos al igual que XML, los dos componentes pueden compartir datos y utilizar esquemas XML para definir la estructura relacional del conjunto de datos. Y, debido a que el formato de serialización del conjunto de datos es XML, los objetos DataSet pueden traspasar fácilmente los servidores de seguridad sin restricciones.

Como hemos visto antes, a la hora de desarrollar módulos de acceso a datos, para cualquier tipo de aplicaciones, tenemos tres opciones: ADO, OLE DB y ADO.NET. Veamos ahora, cada uno de ellos y la evolución de ADO.NET a partir de ADO.NET y ODBC.

4.2.1 ADO.NET como evolución de ADO y OLE DB (ODBC)

Según el encare que le da Microsoft a sus presentaciones de ADO.NET, en general lo presentan como la evolución lógica que debían tener el acceso a dato de ADO y el acceso mediante OLE DB.

ADO

Para aplicaciones escritas en código nativo, ADO proporciona una interfaz de aplicación basada en COM para los proveedores de datos OLE DB. De manera similar a ADO.NET, ADO permite abordar una gran variedad de necesidades de desarrollo, incluida la creación de clientes de bases de datos de aplicaciones para usuarios y objetos comerciales de nivel medio mediante el uso de conexiones directas con datos de bases de datos relacionales y otros tipos de almacenes. Al igual que ADO.NET, ADO puede construir conjuntos de registros de cliente, utilizar conjuntos de registros de correspondencia imprecisa y controlar conjuntos de filas de estructuración de datos de OLE DB.

ADO también admite algunos comportamientos no expuestos a través de ADO.NET, como cursores de servidor con desplazamiento. Sin embargo, como los cursores de servidor utilizan recursos de base de datos, su utilización podría afectar negativamente al rendimiento y escalabilidad de la aplicación. Para transmitir conjuntos de registros ADO a través de servidores de seguridad, es necesario configurar el servidor de seguridad para habilitar la petición de cálculo de referencias de COM, teniendo en cuenta las opciones de seguridad asociadas. El cálculo de referencias de COM también limita los tipos de datos a los definidos por el estándar de COM. También se puede almacenar un conjunto de registros ADO en formato XML y transmitir en su lugar el texto XML.

OLE DB

OLE DB es la interfaz fundamental de programación del sistema para tener acceso a los datos, y constituye la tecnología subyacente de ADO y el origen de datos de ADO.NET. OLE DB es un estándar abierto para tener acceso a todo tipo de datos, tanto relacionales como no relacionales, entre los que se incluyen: ISAM/VSAM de gran sistema (mainframe) y bases de datos jerárquicas, almacenes de sistemas de archivos y de correo electrónico, datos geográficos, gráficos y de texto y objetos comerciales personalizados.

OLE DB proporciona un acceso a los datos coherente y de gran rendimiento, y permite abordar una gran variedad de necesidades de desarrollo, incluida la creación de clientes de bases de datos de aplicaciones para el usuario y objetos comerciales del nivel medio mediante el uso de conexiones directas con datos de bases de datos relacionales y otros tipos de almacenes.

4.2.2 ADO.NET, evolución de la tecnología ADO y la tecnología ODBC:

ADO.NET es una librería que contiene un conjunto de clases que exponen el servicio de acceso a datos para los programadores del entorno .NET. ADO.NET provee un rico conjunto de componentes para crear, compartir y distribuir datos de distintos tipos de aplicaciones. Es una parte integral del .NET Framework, que provee el acceso a datos relacionales, XML, etc. ADO.NET soporta una variedad de necesidades comunes de los desarrolladores, incluyendo la creación de un front-end de cliente de base de datos (para trabajar fuertemente del lado del cliente si es deseado) y facilidades para crear una capa intermedia de acceso a datos, que pueda luego ser usada en aplicaciones locales, herramientas remotas, accedidas desde internet, etc.

ADO.NET es lo más reciente en una extensa línea de tecnologías de acceso a bases de datos que comenzó hace varios años con la interfaz de programación de aplicaciones (API) de la conectividad abierta de base de datos (Open Database Connectivity, ODBC).

Transcurrido este tiempo, han sucedido varias cosas interesantes. Por ejemplo, COM comenzó a trabajar en el terreno de las bases de datos, lo que culminó con OLE DB. Posteriormente, ActiveX® Data Objects (ADO) (más o menos, una versión automatizada de OLE DB) se ha elegido para regir la comunidad Visual Basic® y ASP de los desarrolladores de base de datos basados en Windows®.

ADO.NET representa el sustrato que compondrá la base de las aplicaciones .NET compatibles con datos. A diferencia de ADO, ADO.NET se ha diseñado siguiendo específicamente unas directrices más generales y menos orientadas a la base de datos. ADO.NET es, en pocas palabras una biblioteca que reúne todas las clases que permiten el manejo de datos. Estas clases representan los objetos que contienen datos y que muestran las capacidades normales de las bases de datos: índices, ordenación, vistas. Aunque ADO.NET es la solución definitiva para las aplicaciones de base de datos de .NET, destaca por un diseño global que no se centra tanto en las bases de datos como el modelo ADO. Una prueba de este perfil genérico es que los DataSets o conjuntos de datos, pueden crearse tanto a partir de consultas a la base de datos como a partir de archivos XML. De forma muy simple y similar lo podemos hacer a partir de ambas fuentes de datos y luego, el manejo del conjunto de datos es totalmente desligado del origen de los mismos.

El acceso a los orígenes de datos en ADO.NET se rige a través de proveedores administrados, o DataProviders, como se los llama jerga de .NET Framework. Respecto a las funciones, un proveedor administrado es muy parecido a un proveedor OLE DB, aunque con dos diferencias importantes. Primero, funcionan en el entorno .NET y recuperan y exponen datos a través de clases .NET, como DataReader y DataTable, DataColumn, DataRelation, etc. Segundo, la arquitectura resulta más sencilla, puesto que se ha optimizado para .NET.

Si nos restringimos al acceso a bases de datos, en la actualidad, ADO.NET incorpora dos tipos de proveedores administrados: uno para SQL Server™ 7.0 o posterior y otro para todos los demás proveedores OLE DB que pueda tener instalados. Las clases que se utilizan en ambas situaciones son distintas, aunque siguen una convención de nomenclatura parecida. Los nombres son iguales, excepto por los prefijos. El prefijo es SQL en el primer caso y ADO en el segundo.

Se recomienda utilizar clases SQL para obtener acceso a las tablas de SQL Server, ya que se dirigen directamente a la API interna del servidor de base de datos, omitiendo el nivel intermedio representado por el proveedor OLE DB. Esto claro está, es una ventaja del hecho de que ambas tecnologías sean provistas por un mismo proveedor. Las clases de ADO constituyen una interfaz .NET además de los proveedores OLE DB y utilizan el puente COM Interop para realizar esta tarea.

Las clases de ADO.NET se encuentran en la librería System.Data, y están integradas con las clases de manejo de XML que se encuentran en la librería System.XML.

4.3 Objetivos del diseño de ADO.NET

A medida que la programación de aplicaciones ha evolucionado, las nuevas aplicaciones se han convertido en aplicaciones de correspondencia imprecisa basadas en el modelo de aplicación Web. Las aplicaciones de hoy en día utilizan cada vez más XML para codificar datos que se van a pasar a través de conexiones de red. Las aplicaciones Web usan HTTP para las comunicaciones entre niveles y, por tanto, deben controlar expresamente el mantenimiento del estado de una solicitud a otra. Este nuevo modelo es muy diferente del estilo de programación con conexión y de correspondencia precisa que caracterizaba la época cliente-servidor, en la que una conexión permanecía abierta durante toda la vida del programa y no hacía falta controlar el estado.

ADO.NET supone un modelo de programación totalmente nuevo para el acceso a datos, un modelo basado en .NET Framework. Tomar .NET Framework como base garantizaba que la tecnología de acceso a datos sería uniforme: los componentes compartirían su sistema de tipos, sus modelos de diseño y sus convenciones de nomenclatura.

ADO.NET se diseñó con el propósito de satisfacer las necesidades de este nuevo modelo de programación: arquitectura de datos sin mantener una conexión abierta, estrecha integración con XML, representación común de datos con la posibilidad de combinar datos procedentes de múltiples y variados orígenes, y servicios optimizados para interactuar con una base de datos, todo ello nativo de .NET Framework. En definitiva, se podría decir que a la hora de crear ADO.NET, Microsoft se propuso los siguientes objetivos de diseño.

Aprovechar los conocimientos actuales de ADO

El diseño de ADO.NET satisface muchos de los requisitos del modelo de desarrollo de aplicaciones de hoy en día. Al mismo tiempo, el modelo de programación sigue siendo lo más parecido posible a ADO, de manera que los programadores actuales de ADO no tienen que empezar a aprender desde cero una tecnología de acceso a datos totalmente nueva. ADO.NET forma parte intrínseca de .NET Framework sin que al programador de ADO le parezca completamente extraña.

ADO.NET coexiste con ADO. Aunque la mayoría de las nuevas aplicaciones basadas en .NET se escribirán mediante ADO.NET, ADO sigue estando disponible para el programador de .NET a través de los servicios de interoperabilidad COM de .NET.

Admitir el modelo de programación N-Tier

ADO.NET proporciona compatibilidad de primera clase con el entorno de programación n-tier sin mantener una conexión abierta para el que están escritas muchas aplicaciones nuevas. La idea de trabajar con un conjunto de datos sin mantener una conexión abierta se ha convertido en un objetivo del modelo de programación. La solución de ADO.NET para la programación n-tier es el DataSet. El DataSet abstrae totalmente el acceso a datos.

Integrar la compatibilidad con XML

En ADO, XML era simplemente un formato de entrada y salida. No obstante, en ADO.NET, XML es el formato de datos que proporciona los medios para manipular, reorganizar, compartir y transferir datos. Todo grupo de datos incluidos en DataSet, independientemente del origen, se puede manipular a través de un modelo de programación de dos caras. Puede obtener acceso de forma intercambiable a la información, fila tras fila y, asimismo, puede seguir una ruta jerárquica, no secuencial, dirigida por el modelo de objeto del documento XML.

Un DataSet lee y escribe datos y esquemas como documentos XML. Los datos y los esquemas se pueden transportar en HTTP y utilizar en cualquier plataforma que comprenda XML. Los mismos datos se pueden

procesar a través de distintos esquemas (XSLT encuentra la forma de realizarlo) en momentos distintos. El método ReadXmlSchema se utiliza para generar esquemas. Un esquema XML incluye una descripción de las tablas en el juego de datos, así como las relaciones y restricciones de éstas. Debe realizar esta operación antes de solicitar que el método ReadXmlData rellene el DataSet.

XML y el acceso a datos están estrechamente relacionados: XML consiste en codificar datos y el acceso a datos consiste cada vez más en XML. .NET Framework no sólo admite los estándares Web, sino que está basado totalmente en ellos.

La compatibilidad con XML está integrada en los cimientos de ADO.NET. Las clases de XML incluidas en .NET Framework y ADO.NET forman parte de la misma arquitectura: están integradas en muchos niveles. Ya no es necesario elegir entre el conjunto de servicios de acceso a datos y los correspondientes servicios de XML; la capacidad para cruzar de uno a otro es inherente al diseño de ambos. ADO.NET aprovecha la eficacia de XML para proporcionar acceso a datos sin mantener una conexión abierta. ADO.NET fue diseñado teniendo en cuenta las clases de XML incluidas en .NET Framework: ambos son componentes de una única arquitectura.

ADO.NET y las clases de XML incluidas en .NET Framework convergen en el objeto DataSet. El DataSet se puede llenar con datos procedentes de un origen XML, ya sea éste un archivo o una secuencia XML. El DataSet se puede escribir como XML compatible con el del Consorcio World Wide Web (W3C), incluyendo su esquema como esquema XSD (Lenguaje de definición de esquemas XML), independientemente del origen de los datos incluidos en el DataSet. Puesto que el formato nativo de serialización del DataSet es XML, es un medio excelente para mover datos de un nivel a otro, por lo que el DataSet es idóneo para utilizar datos y contextos de esquemas de interacción remota desde y hacia un servicio Web XML.

Además de cargar datos de XML, los conjuntos de datos se pueden rellenar con y mantener los cambios de datos de SQL Server así como de orígenes de datos expuestos a través de OLE DB.

Una de las características principales de los conjuntos de datos es que permiten el acceso y la manipulación de los datos incluidos en un conjunto de datos local de dos formas diferentes:

- **Como tablas de una base de datos relacional:** Un conjunto de datos puede contener una tabla o una colección de tablas. Un aspecto importante del conjunto de datos es que realiza un seguimiento de las relaciones entre las tablas que contiene, como si se tratase de un almacén de datos relacionales en memoria.
- **Como estructuras XML (Lenguaje de marcado extensible):** También es posible el acceso a un conjunto de datos en forma de datos XML. Existen métodos para leer y escribir datos como XML y para leer y escribir la estructura del conjunto de datos como un esquema XML. Adicionalmente, se puede asociar un XmlDocument a un conjunto de datos para ver, consultar y modificar los datos como XML de manera simultánea.

El DataSet también se puede sincronizar con un XmlDocument para proporcionar acceso jerárquico y relacional a datos en tiempo real.

Arquitectura de ADO.NET.

El acceso a datos estuvo tradicionalmente basado en una arquitectura de dos capas orientada a trabajar con conexión. Dado que incrementalmente se ha ido introduciendo la utilización de arquitectura multi-capa, se ha hecho necesario orientar el desarrollo hacia aplicaciones desconectadas para proveer mayor escalabilidad a las aplicaciones.

ADO.NET utiliza el poder de XML, para brindar acceso desconectado a datos. Tanto ADO.NET como las librerías de manejo de XML fueron diseñadas en paralelo, paso a paso, y de forma muy relacionada en el .NET Framework, siendo ambos componentes de una misma arquitectura. Las clases de ADO.NET y XML.NET convergen en el objeto DataSet. El DataSet o conjunto de datos es una estructura relacional desconectada del origen de datos, que se carga a partir de éste y luego se puede administrar de forma independiente del origen. Al DataSet se le pueden eliminar, agregar, cambiar datos, aparte de la consulta, para luego, si es deseado, impactar los cambios en la base de datos.

Componentes de ADO.NET

Los componentes de ADO.NET fueron especialmente diseñados para facilitar el acceso a datos y su manipulación. Los dos componentes centrales, necesarios para satisfacer este principio son: el DataSet, y los proveedores de datos del Framework (.NET Framework data providers), un conjunto de clases representantes de los distintos conceptos involucrados en un acceso a datos, como ser conexiones (Connection), comandos SQL (Command), objetos utilizados para leer datos (DataReader), o para escribir en base de datos (DataAdapter).

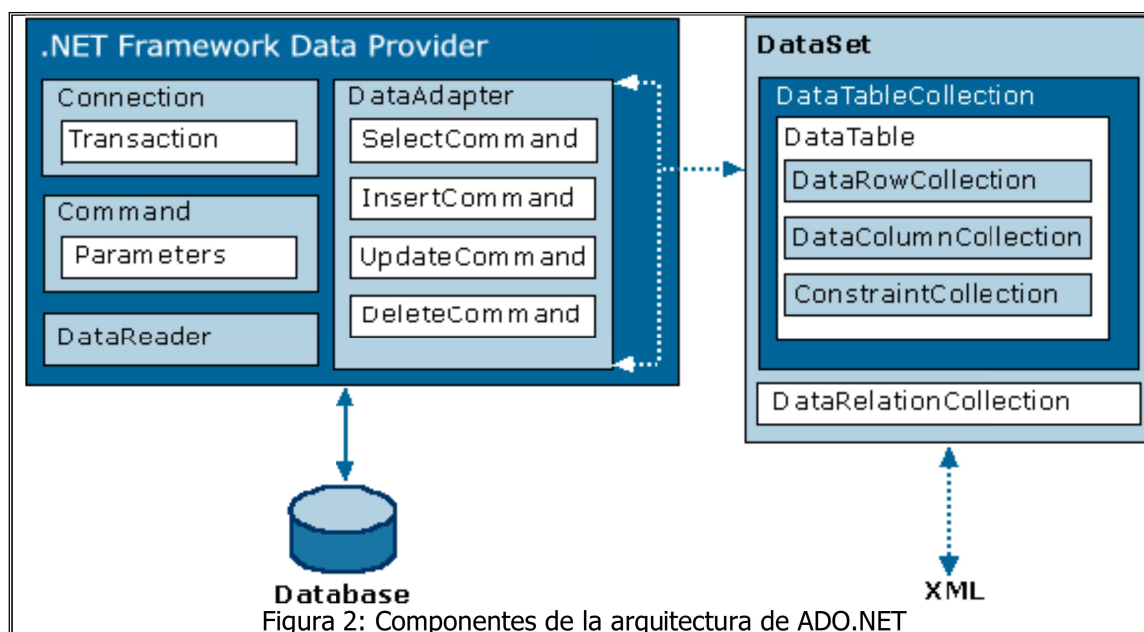


Figura 2: Componentes de la arquitectura de ADO.NET

El DataSet es el objeto central de la arquitectura desconectada de ADO.NET. Está implícitamente diseñado para encapsular el acceso a datos, independientemente del origen de los mismos. Por otro lado los objetos que encontramos entre los proveedores de datos, resuelven todas las operaciones de conexión y acceso a base de datos. **El DataAdapter**, si bien está incluido en el segundo componente, **resuelve todo el mapeo entre los objetos de acceso físico a la base** (conexiones, comandos, parámetros, transacciones, etc.) **y la estructura relacional desconectada** (el DataSet). Básicamente tenemos el mundo conectado (Adaptadores), el mundo del trabajo desconectado (DataSet) y el puente entre ellos (DataAdapter). Es posible escribir proveedores de datos de .NET para cualquier origen de datos. .NET Framework incluye dos proveedores de datos de .NET: el proveedor de datos .NET de SQL Server y el proveedor de datos .NET de OLE DB.

4.4 DataSet de ADO.NET

El objeto DataSet es esencial para admitir escenarios de datos distribuidos de ADO.NET sin mantener una conexión. El DataSet es una representación residente en memoria de datos que proporciona un modelo de programación relacional coherente independientemente del origen de datos. Se puede utilizar con múltiples y distintos orígenes de datos, con datos XML o para administrar datos locales de la aplicación. El DataSet representa un conjunto completo de datos entre los que se incluyen tablas relacionadas, restricciones y relaciones entre las tablas, claves primarias, y demás conceptos implícitos en cualquier estructura de almacenamiento relacional. Por otro lado, proporciona una serie de potencialidades extra, que mejoran la administración de los datos. En la siguiente ilustración, se muestra el modelo del objeto DataSet.

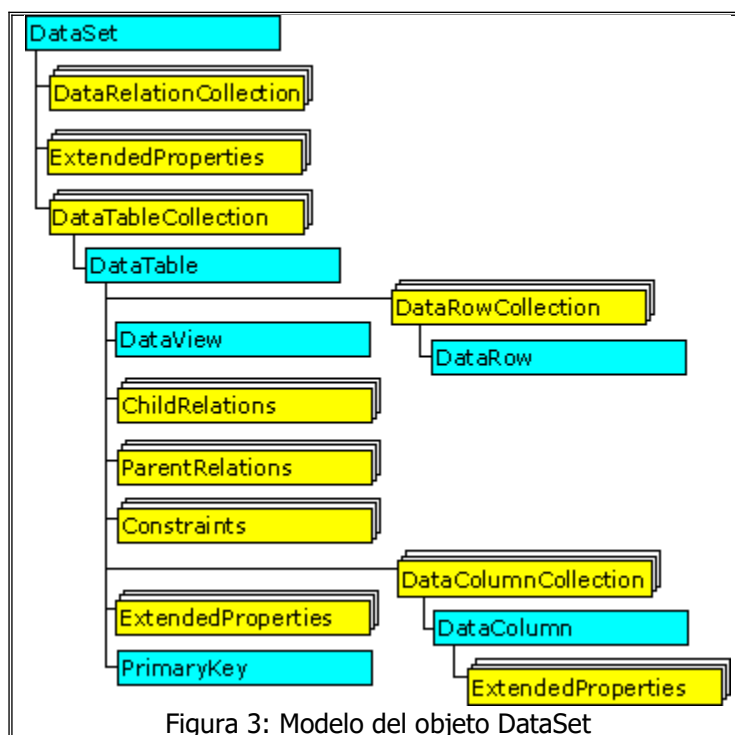


Figura 3: Modelo del objeto DataSet

Los métodos y objetos contenidos en un DataSet son coherentes con los del modelo de base de datos relacional. El DataSet también puede persistir y volver a cargar su contenido como XML y su esquema como esquema XSD (Lenguaje de definición de esquemas XML).

La DataTableCollection

Un DataSet de ADO.NET contiene una colección de cero o más tablas representadas por objetos DataTable. La DataTableCollection contiene todos los objetos DataTable de un DataSet.

Un DataTable se define en el espacio de nombres System.Data y representa una única tabla de datos residentes en memoria. Contiene una colección de columnas representadas por una DataColumnCollection y restricciones representadas por una ConstraintCollection que, juntas, definen el esquema de la tabla. Un DataTable también contiene una colección de filas representadas por la DataRowCollection, que contiene los datos de la tabla. Junto con su estado actual, un DataRow conserva tanto la versión original como la actual para identificar los cambios realizados en los valores almacenados en la fila.

La DataRelationCollection

Un DataSet contiene relaciones en su objeto DataRelationCollection. Una relación, representada por el objeto DataRelation, asocia las filas de un DataTable con las filas de otro DataTable. Es análogo a una ruta de unión que podría existir entre las columnas de claves externas y principales en una base de datos relacional. Un DataRelation identifica columnas coincidentes en dos tablas de un DataSet.

Las relaciones permiten pasar de una tabla a otra dentro de un mismo DataSet. Los elementos esenciales de un DataRelation son el nombre de la relación, el nombre de las tablas relacionadas y las columnas relacionadas de cada tabla. Se pueden establecer relaciones con más de una columna por tabla, para lo que debe especificar una selección de objetos DataColumn como columnas clave. Cuando se agrega una relación al DataRelationCollection, se puede agregar también un UniqueKeyConstraint y un ForeignKeyConstraint para imponer restricciones de integridad cuando se realicen cambios en los valores de las columnas relacionadas.

ExtendedProperties

El DataSet (así como el DataTable y el DataColumn) tiene una propiedad ExtendedProperties. ExtendedProperties es un PropertyCollection en el que se puede colocar información personalizada, como la instrucción SELECT que sirve para generar el conjunto de resultados, o una marca de fecha y hora de cuándo se generaron los datos. La colección ExtendedProperties se conserva con la información del esquema del DataSet (así como el DataTable y el DataColumn). Básicamente es un Hashtable al que se pueden agregar propiedades y en otro momento consultarlas por nombre. Al serializar un DataSet a XML, las ExtendedProperties se serializan con el resto del objeto y se recuperan exactamente iguales, cuando se deserialice el objeto.

Con todo esto, parece claro que el objeto DataSet tiene la gran mayoría, por no decir todas las propiedades que uno podría buscar en una estructura de almacenamiento de información.

4.4.1 Proveedores de datos de .NET

Un proveedor de datos de .NET sirve para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Esos resultados se procesan directamente o se colocan en un DataSet de ADO.NET con el fin de exponerlos al usuario para un propósito específico, junto con datos de varios orígenes, o de utilizarlos de forma remota entre niveles. El diseño del proveedor de datos de .NET hace que sea ligero, de manera que cree un nivel mínimo entre el origen de datos y su código, con lo que aumenta el rendimiento sin sacrificar la funcionalidad.

En la siguiente tabla se describen los cuatro objetos centrales que constituyen un proveedor de datos:

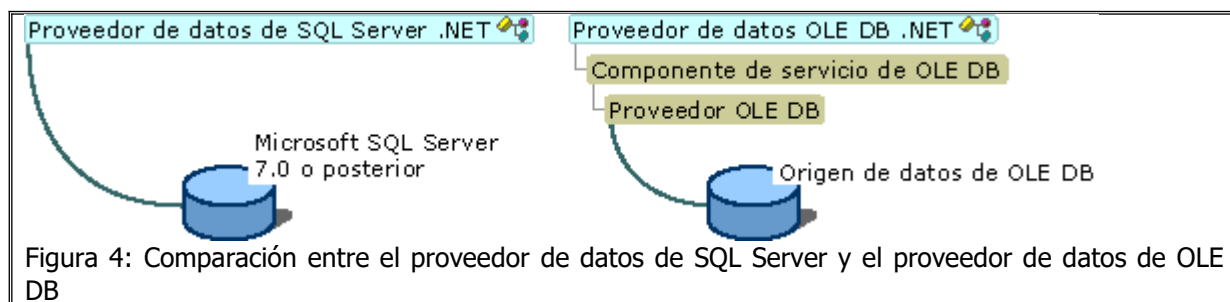
Objeto	Descripción
Connection	Establece una conexión a un origen de datos determinado.
Command	Ejecuta un comando en un origen de datos. Expone Parámetros y puede ejecutarse en el ámbito de una Transacción de una Conexión .
DataReader	Lee una secuencia de datos de sólo avance y sólo lectura desde un origen de datos.
DataAdapter	Llena un DataSet y realiza las actualizaciones necesarias en el origen de datos.

.NET Framework incluye el proveedor de datos .NET de SQL Server (para Microsoft SQL Server 7.0 o posterior) y el proveedor de datos .NET de OLE DB.

Para obtener un proveedor de datos .NET de ODBC (Open Database Connectivity, Conectividad abierta de base de datos) puede descargarlo de forma individual de sitios de Microsoft. La descarga incluye documentación de las clases que componen el proveedor de datos .NET de ODBC. Sin embargo, su implementación tiene la misma arquitectura que la de los proveedores de datos .NET de SQL Server y OLE DB.

El proveedor de datos .NET de SQL Server

El proveedor de datos de .NET de SQL Server utiliza su propio protocolo para establecer comunicación con SQL Server. Es ligero y presenta un buen rendimiento porque está optimizado para tener acceso a SQL Server directamente, sin agregar una capa OLE DB u ODBC. En la siguiente ilustración se compara el proveedor de datos de .NET de SQL Server con el proveedor de datos de .NET de OLE DB. El proveedor de datos de .NET de OLE DB se comunica con un origen de datos OLE DB tanto a través del componente de servicio OLE DB, que proporciona agrupación de conexiones y servicios de transacción, como del proveedor OLE DB correspondiente al origen de datos.



Para utilizar el proveedor de datos de .NET de SQL Server, debe tener acceso a Microsoft SQL Server 7.0 o posterior. Las clases del proveedor de datos de .NET de SQL Server están ubicadas en el espacio de nombres System.Data.SqlClient. Para las versiones anteriores de Microsoft SQL Server, use el proveedor de datos de .NET de OLE DB con el proveedor OLE DB de SQL Server (SQLOLEDB).

El proveedor de datos .NET de OLE DB.

El proveedor de datos de .NET de OLE DB utiliza OLE DB nativo para permitir el acceso a datos mediante la interoperabilidad COM. El proveedor de datos de .NET de OLE DB admite tanto transacciones manuales como automáticas. Para transacciones automáticas, el proveedor de datos de .NET de OLE DB se inscribe automáticamente en una transacción y obtiene los detalles de la misma a través de los servicios de componentes de Windows 2000.

Para utilizar el proveedor de datos de .NET de OLE DB, debe usar un proveedor OLE DB que sea compatible con las interfaces OLE DB enumeradas en [Interfaces OLE DB que utiliza el proveedor de datos OLE DB .NET](#).

En la siguiente tabla se muestra los proveedores que se han probado con ADO.NET.

Controlador	Proveedor
SQLOLEDB	Proveedor OLE DB para SQL Server de Microsoft
MSDAORA	Proveedor OLE DB para Oracle de Microsoft

Microsoft.Jet.OLEDB.4.0 Proveedor OLE DB para Microsoft Jet

Las clases del proveedor de datos de .NET de OLE DB están ubicadas en el espacio de nombres System.Data.OleDb. En el siguiente ejemplo de código se muestra cómo puede incluir el espacio de nombres System.Data.OleDb en sus aplicaciones. El proveedor de datos de .NET de OLE DB requiere la instalación de MDAC 2.6 o posterior. El mismo se instala automáticamente al instalar el .NET Framework.

Elegir un proveedor de datos de .NET

Dependiendo del diseño y del origen de datos de la aplicación a desarrollar, la elección del proveedor de datos de .NET puede mejorar el rendimiento, las características y la integridad de la misma. En la siguiente tabla se describen las ventajas y las limitaciones de cada proveedor de datos de .NET.

Proveedor	Notas
Proveedor de datos de .NET de SQL Server	Recomendado para aplicaciones de nivel medio que utilizan Microsoft SQL Server 7.0 o posterior. Recomendado para aplicaciones de un único nivel que utilizan Microsoft Data Engine (MSDE) o Microsoft SQL Server 7.0 o posterior. Recomendado en lugar de utilizar el proveedor OLE DB para SQL Server (SQLOLEDB) con el proveedor de datos de .NET de OLE DB. Para la versión 6.5 y anteriores de Microsoft SQL Server, debe utilizar el proveedor OLE DB para SQL Server con el proveedor de datos de .NET de OLE DB. Recomendado para aplicaciones de nivel medio que utilizan Microsoft SQL Server 6.5 o anteriores, o cualquier proveedor OLE DB que admita las interfaces OLE DB enumeradas en Interfaces OLE DB que utiliza el proveedor de datos OLE DB .NET.
Proveedor de datos de .NET de OLE DB	Recomendado para aplicaciones de un único nivel que utilizan bases de datos de Microsoft Access. No se recomienda el uso de una base de datos de Microsoft Access para una aplicación de nivel medio. La compatibilidad con el proveedor OLE DB para ODBC (MSDASQL) está deshabilitada. En http://msdn.microsoft.com/downloads se puede descargar un proveedor de datos de .NET de ODBC para tener acceso a orígenes de datos de ODBC (Conectividad abierta de bases de datos).

4.5 Adaptadores de datos de .NET

DataSet de ADO.NET es una representación residente en memoria de datos que proporciona un modelo de programación relacional coherente e independiente del origen de los datos. DataSet representa un conjunto completo de datos que incluye tablas y restricciones, así como relaciones entre ellas. Dado que DataSet es independiente del origen de datos, puede incluir datos locales de la aplicación, así como datos de otros muchos orígenes. La interacción con los orígenes de datos existentes se controla mediante el adaptador de datos o **DataAdapter**.

Cada proveedor de datos .NET incluido en .NET Framework tiene un objeto DataAdapter: el proveedor de datos .NET de OLE DB incluye un objeto OleDbDataAdapter, mientras que el proveedor de datos .NET de SQL Server incluye un objeto SqlDataAdapter. Para recuperar datos de un origen de datos y llenar las tablas de un DataSet, se utiliza un DataAdapter. El DataAdapter sirve también para reflejar en el origen de datos los cambios efectuados en el DataSet. El DataAdapter utiliza el objeto Connection del proveedor de datos .NET para establecer una conexión con un origen de datos, así como objetos Command para recuperar datos de un origen de datos y reflejar en él los cambios que se efectúen.

La propiedad **SelectCommand** del DataAdapter es un objeto Command que recupera datos del origen de datos.

Las propiedades **InsertCommand**, **UpdateCommand** y **DeleteCommand** de DataAdapter son objetos Command que permiten administrar las actualizaciones de los datos en el origen de datos para reflejar las modificaciones efectuadas en el DataSet.

El método **Fill** del DataAdapter se usa para llenar un DataSet con los resultados de la propiedad SelectCommand del DataAdapter. El método Fill acepta como argumentos un DataSet que se debe llenar y un objeto DataTable, o su nombre, que se debe llenar con las filas que devuelve SelectCommand.

El método Fill utiliza el objeto DataReader de forma implícita para devolver los nombres y tipos de columna usados para crear las tablas de DataSet, así como los datos para llenar las filas de las tablas de DataSet. Las tablas y columnas sólo se crean cuando no existen, en caso contrario, Fill usa el esquema existente de DataSet. Los tipos de columna se crean como tipos de .NET Framework. No se crean claves principales a menos que existan en el origen de datos y se haya dado el valor MissingSchemaAction.AddWithKey a DataAdapter.MissingSchemaAction. Si el método Fill determina que una tabla tiene clave principal, sobrescribe los datos del DataSet con los del origen de datos en aquellas filas en las que los valores de la columna de clave principal coincidan con los de la fila que devuelve el origen de datos. Si no se detecta ninguna clave principal, los datos se agregan a continuación de los ya existentes en las tablas del DataSet. Fill utiliza cualquier TableMappings que pueda existir al llenar el DataSet. La propiedad TableMapping del DataAdapter es una colección de mapeos entre las tablas y columnas del origen de datos y tablas y columnas del DataSet. Si no existe mapeo para determinada columna, DataAdapter la mapea a una columna de igual nombre en el DataSet, y en caso que no exista la crea.

Llenar un DataSet desde múltiples DataAdapter

Con un DataSet se puede usar un número ilimitado de DataAdapters. Cada DataAdapter se puede utilizar para llenar uno o varios objetos DataTable y para reflejar en el origen de datos correspondiente las actualizaciones que sean necesarias. Se pueden agregar de forma local objetos DataRelation y Constraint al DataSet, de manera que se pueden relacionar datos procedentes de múltiples orígenes distintos. Por ejemplo, un DataSet puede contener datos de una base de datos de Microsoft SQL Server, una base de datos de IBM DB2 expuesta mediante OLE DB y de un origen de datos que genera secuencias XML. Para ocuparse de la comunicación con cada origen de datos se pueden usar uno o varios objetos DataAdapter. En general, según varios documentos observados la experiencia indica que lo más simple y seguro es generar un DataAdapter específico por cada DataTabe del DataSet.

4.6 LOS ESPACIOS DE NOMBRES DE ADO.NET

ADO.NET se encuentra en la biblioteca System.Data.dll, y ofrece clases en cinco espacios de nombres bien diferenciados que explico brevemente a continuación:

System.Data

Es el espacio de nombres primario. Dentro de este espacio de nombres tenemos un conjunto de clases que representan, digamos, una base de datos virtual, tablas, filas, columnas, relaciones, etc. Sin embargo, ninguna de estas clases ofrece conexión alguna con un origen de datos, sino que simplemente representan los datos en sí mismos.

System.Data.Common

Ofrece clases comunes entre distintos orígenes de datos.

System.Data.OleDb

Contiene una serie de clases que nos permiten conectarnos con cualquier origen de datos e interactuar con él al tiempo que sirven de "intermediarios" entre el origen de datos y las clases del espacio de nombres System.Data que, como decíamos, no tienen conexión alguna con dicho origen de datos. Las clases de System.Data.OleDb usan OLEDB como tecnología subyacente.

System.Data.SqlClient

Contiene clases que permiten interactuar con orígenes de datos SQL Server de un modo mucho más directo que OLEDB, mejorando el rendimiento para este tipo de origen de datos. Por lo tanto, solamente se pueden utilizar para acceder a bases de datos de SQL Server. El uso de sus clases es prácticamente equivalente al de las que se encuentran en System.Data.OleDb.

System.Data.SqlTypes

Este espacio de nombres ofrece los tipos primitivos que usa SQL Server. Obviamente, aunque se pueden usar los tipos equivalentes del CTS, los que se incluyen en este espacio de nombres están optimizados para trabajar con SQL Server.

4.7 Conceptos interesantes al enfrentarse a ADO.NET

Sincronizar DataSet con XmlDocument

El DataSet de ADO.NET proporciona una representación relacional de datos. Para el acceso a datos jerárquicos puede utilizar las clases XML disponibles en .NET Framework. Históricamente, estas dos representaciones de datos se han utilizado independientemente. Sin embargo, .NET Framework permite el acceso sincrónico en tiempo real, tanto a la representación relacional como a la representación jerárquica de los datos mediante los objetos DataSet y XmlDocument, respectivamente.

Cuando un DataSet se sincroniza con un XmlDocument, ambos objetos trabajan con un único conjunto de datos. Esto significa que si se realiza un cambio en el DataSet, dicho cambio quedará reflejado en el XmlDocument, y viceversa. La relación entre el DataSet y el XmlDocument ofrece una gran flexibilidad, ya que permite que una única aplicación, utilizando un único conjunto de datos, tenga acceso a todo el conjunto de servicios existentes en el DataSet (como controles Web Forms y Windows Forms, y diseñadores de Visual Studio .NET), así como al conjunto de servicios XML, incluyendo XSL (Extensible Stylesheet Language), XSLT (XSL Transformations) y XPath (XML Path Language). No tiene que elegir el conjunto de servicios a los que desea dirigir la aplicación, ya que ambos están disponibles.

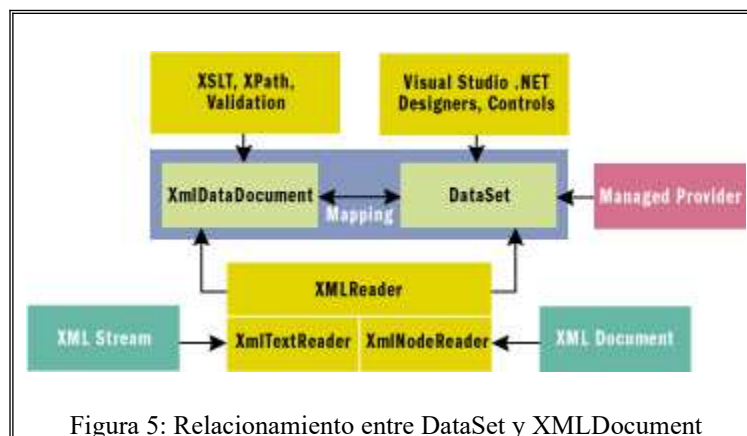


Figura 5: Relacionamiento entre DataSet y XmlDocument

Hay varias formas de sincronizar un DataSet con un XmlDocument. Se puede:

- Llenar un DataSet con el esquema (estructura relacional) y datos y, después, sincronizarlo con un nuevo XmlDocument. Esto ofrece una vista jerárquica de los datos relacionales existentes.
- Llenar un DataSet con el esquema únicamente (como un DataSet con establecimiento inflexible de tipos), sincronizarlo con un XmlDocument y cargar el XmlDocument desde un documento XML. Esto ofrece una vista relacional de los datos jerárquicos existentes. Los nombres de tabla y de columna del esquema del DataSet deben coincidir con los nombres de los elementos XML con los que desea sincronizarlos. La coincidencia distingue mayúsculas de minúsculas.

Hay que tener en cuenta que en el esquema del DataSet sólo es necesario que coincidan los elementos XML que desee exponer en la vista relacional. De esta forma puede tener un documento XML muy grande y una "ventana" relacional muy pequeña de ese documento. El XmlDocument conservará todo el documento XML, incluso aunque el DataSet sólo exponga una pequeña parte del mismo

- Crear un nuevo XmlDocument y cargarlo desde un documento XML, y después tener acceso a la vista relacional de los datos mediante la propiedad DataSet del XmlDocument. En este caso hay que establecer el esquema del DataSet antes de poder ver cualquiera de los datos del XmlDocument mediante el DataSet. Una vez más, los nombres de tabla y de columna del esquema del DataSet deben coincidir con los nombres de los elementos XML con los que desea sincronizarlos. La coincidencia distingue mayúsculas de minúsculas.

Otra ventaja de sincronizar un XmlDocument con un DataSet es que se conserva la fidelidad de un documento XML. Si el DataSet se llena desde un documento XML mediante ReadXml, cuando se vuelvan a escribir los datos como un documento XML mediante WriteXml pueden diferir considerablemente con respecto al documento XML original. Esto se debe a que el DataSet no conserva el formato, como el espacio en blanco, ni la información jerárquica, como el orden de los elementos, del documento XML. El DataSet tampoco contiene elementos del documento XML que se pasaron por alto porque no coincidían con el esquema del DataSet. La sincronización de un XmlDocument con un DataSet permite conservar el formato y la estructura jerárquica de los elementos del documento XML original en el XmlDocument, mientras que el DataSet sólo contiene datos e información de esquema apropiados para el DataSet.

Al sincronizar un DataSet con un XmlDocument, los resultados obtenidos pueden diferir dependiendo de si los objetos DataRelation están o no anidados.

4.7.1 Elegir entre DataReader y DataSet:

Volviendo al diagrama original de ADO.NET, vemos que tenemos del lado del acceso conectado, el objeto DataReader. Este objeto permite acceder de forma rápida, secuencial, a una fuente de datos. La performance es claramente superior a la obtenida cuando se accede a una fuente de datos mediante un DataSet, dado que evitamos etapas del proceso de transformación (en pocas palabras, evitamos utilizar el adaptador de datos, para llenar el contenido del DataSet). Cuando utilizamos el DataSet, lo hacemos por medio de un DataAdapter, que lee de la base mediante un DataReader y transforma la información, cargandola en el DataSet. Esta es la demostración más clara de la pérdida de performance. Por otro lado, el tener los datos en un DataSet, brinda una flexibilidad en su administración que no brinda la lectura directa de un DataReader.

En resumen, hay una serie de puntos a tomar en cuenta a la hora de decidir si es o no conveniente el uso de los objetos DataSet, pero está totalmente ligado al caso particular de aplicación. En general, es mejor utilizar un DataSet cuando se le quiere dar alguno de los siguientes usos:

- Acceder datos de forma remota, mediante el intercambio de información en formato XML.
- Acceder a XML Web Services. En el 90% de los casos prácticos, un servicio web realiza cierto trabajo y devuelve una cierta información. Si cargamos esta información en un DataSet, la serialización es totalmente transparente al programador. Del lado del servidor, podemos sencillamente devolver el DataSet, mientras que del lado del cliente podemos asignar el resultado de la llamada al servicio a uno de estos objetos, resolviendo el Framework, la serialización y deserialización subyacentes a este proceso.
- El DataSet es una herramienta sumamente útil cuando estamos atacando un problema de integración de datos. Por ejemplo, debemos desarrollar una aplicación que toma datos de distintos orígenes, los relaciona de determinada forma y los vuelca a otra aplicación que sigue con el proceso. Una forma muy sencilla de hacerlo sería crear un DataSet que tenga las tablas de un origen de datos, las tablas del otro, agregarle las tablas que generan la relación, generar las relaciones propiamente dichas, como registro de esas tablas y luego volcar el contenido del DataSet a un documento XML. Este proceso genera un XML, que comienza con el esquema del DataSet y luego contiene los datos. Si en otro proceso del negocio creamos un DataSet a partir del documento XML generado, obtenemos exactamente lo que teníamos antes, manteniendo tablas, relaciones, claves primarias, restricciones de unicidad, claves foráneas y, por supuesto, los datos en sí mismos.
- Proveer una vista relacional de un conjunto de datos que sigue una estructura jerárquica expresada en un documento XML.
- Cuando necesitamos acceder a la fuente de datos y luego realizar operativas pesadas sobre la información obtenida antes de presentarla al usuario puede ser muy conveniente hacerlo mediante la utilización de DataSets, hacer la consulta, desconectarse del origen de datos y luego hacer la operativa sobre el objeto desconectado, liberando así más rápidamente el acceso a la fuente.

Si nuestra aplicación no requiere estas ventajas provistas por el DataSet, probablemente sea más eficiente trabajar directamente con el lector de datos (DataReader).

5 INTRODUCCIÓN A ASP.NET

Internet es la red de redes. Su evolución fue y es vertiginosa. El principal protocolo que utiliza es HTTP.

El contenido en su primera etapa era accedido mediante **páginas HTML estáticas**. HTML es un lenguaje de marcado. Estas marcas o "TAGs" definen la presentación de la información, por ejemplo, el color, la tipografía o si se va a usar una tabla para ordenar la presentación de la información.

Luego se incorporaron lenguajes como Java o VBScript que permitieron poner código en la página HTML. Este código era accedido por el cliente y **se ejecutaba en el equipo del cliente**. Esto significó que las páginas tuvieran un contenido dinámico que podía cambiar, por ejemplo, una animación en el encabezado de una página. El siguiente paso fue el poner **código del lado del servidor** que provee y administra las páginas. La tecnología de Microsoft para esto fue ASP ejecutándose en el servidor "Internet Information Services".

Las **páginas ASP** permiten mezclar las etiquetas HTML con código como Java o VBScript. Bajo este esquema cuando un usuario accede a las páginas HTML o ASP puede disparar la ejecución de código que se encuentra en páginas ASP. Esta da respuesta al cliente como una página HTML. Una ventaja del código del lado del servidor en las páginas ASP es la seguridad ya que el usuario no accede al código. Una desventaja es el desempeño ya que el código se compila en el momento de ser accedido. El siguiente paso de la evolución es **ASP.NET**.

Que es ASP .NET?

ASP.NET es la tecnología que permite hacer **formularios Web y Servicios Web**. Una aplicación Web ASP.NET puede estar formada por varios formularios Web.

- Es el sucesor de las páginas ASP
- Son páginas dinámicas que pueden acceder a recursos del servidor
- Permite la ejecución de la página web del lado del servidor
- Permite hacer servicios web
- Es independiente del explorador de internet
- Es independiente del lenguaje de programación

El **formulario Web** es una página dinámica que puede acceder a los recursos del servidor o a otros servidores. Por ejemplo, una página ASP.NET puede ejecutar código para acceder a SQL Server y luego armar una respuesta al usuario como código HTML.

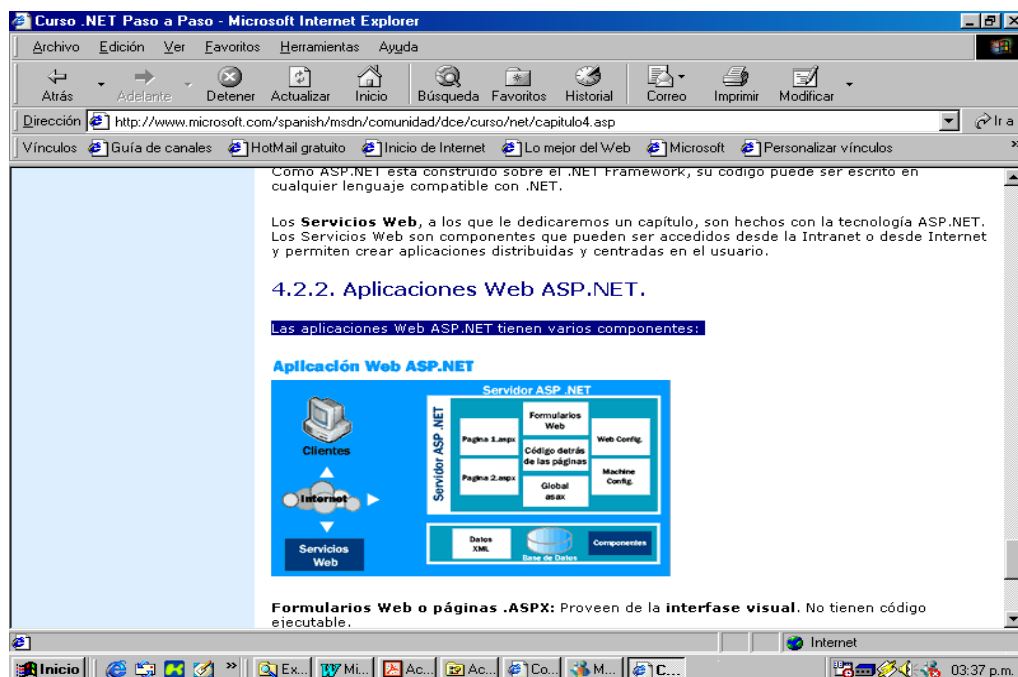
Como el código se ejecuta del lado del servidor, la respuesta puede adaptarse al Explorador de Internet del usuario.

Como ASP.NET esta construido sobre el .NET Framework, su código puede ser escrito en cualquier lenguaje compatible con .NET.

En este diagrama se muestra como es la interacción entre las páginas asp.net con el servidor web (IIS) y con el Framework



Las aplicaciones Web ASP.NET tienen varios componentes:



Formularios Web o páginas .ASPX: Proveen de la **interfase visual**. No tienen código ejecutable.

Páginas de código en archivo aparte: Están asociadas con cada formulario y son las que **proveen del código** ejecutable. A diferencia de las páginas ASP con la tecnología anterior, no se mezcla código y etiquetas en la misma página.

Archivos de configuración: Son archivos que permiten configurar la aplicación, por ejemplo el archivo **web.config** y el servidor, por ejemplo el archivo **machine.config**.

Global.asax: Es un archivo que contiene código. Este código responde a eventos que se disparan en la aplicación Web.

Enlaces a Servicios Web XML: Permite a la aplicación Web transferir datos XML desde y hacia Servicios Web.

Conectividad a Bases de Datos: Permite a la aplicación Web transferir datos desde y hacia distintas Bases de datos.

Caching: Permite a la aplicación Web devolver formularios Web más rápidamente después del primer acceso.

La principal referencia de ASP.NET la tenemos en <http://www.asp.net/>, un portal de Microsoft donde se nos describe la tecnología y sus ventajas. Junto con los motivos para utilizar ASP.NET, tenemos más de 900 ejemplos que nos pueden ayudar a conocer el entorno de forma rápida, junto con tutoriales más detallados para conocer con detalle ciertos aspectos de la plataforma.

La cantidad de artículos, libros y portales dedicados a ASP.NET es bastante importante lo que muestra que ya es una tecnología que poco a poco se va implantando.

ASP.NET es una plataforma que se integra con IIS y que permite que se accedan a páginas con extensión ".aspx" las cuales pueden tener junto al HTML, código en diferentes lenguajes de programación, aunque sólo se debe de utilizar uno por página.

Para el lector familiarizado con PHP, Zope o J2EE, la idea es la misma: facilitar el desarrollo de aplicaciones con interfaz web. Para ello es común montar una infraestructura de librerías y servidores detrás del servidor web. El servidor web recibe la petición y si es de una página activa, terminada en ".aspx", le pasa la petición a la arquitectura ASP.NET.

ASP.NET ofrece varias ventajas importantes sobre los modelos previos de desarrollo para Internet :

Mejor Eficiencia

ASP.NET corre código compilado sobre el entorno NGWS en el servidor. Distinto a sus predecesores interpretados, ASP.NET usa amarres tempranos ("early binding"), así como compilación justo a tiempo ("just-in-time compilation"), optimización nativa, y servicios de caché, sin configuración adicional. Para los desarrolladores, esto significa eficiencia dramáticamente superior antes de escribir la primera línea de código.

Herramientas superiores de desarrollo

ASP.NET tiene una "caja de herramientas" rica: el ambiente de desarrollo integrado de Visual Studio.NET. La edición WYSIWYG, la creación de controles mediante "drag-and-drop", y la publicación automática son varias ventajas.

Poder y Flexibilidad

Porque ASP.NET está basado en el Entorno Común de Ejecución de Lenguajes (Common Language Runtime, o "CLR"), el poder y la flexibilidad de la plataforma completa está disponible para los desarrolladores. Las librerías de Clases del CLR, la Mensajería, y las soluciones de Acceso a Datos, son accesibles al través del Internet. ASP.NET permite el uso de una gran variedad de lenguajes de programación y, por tanto, usted puede escoger el mejor lenguaje para su aplicación, o particionar su aplicación en varios lenguajes. Mas aún, la interoperabilidad del CLR garantiza que su inversión en el desarrollo de aplicaciones COM es preservada cuando se migra a ASP.NET.

Simplicidad

ASP.NET hace fácil el ejecutar tareas comunes, desde el simple envío de un formulario o la autenticación de un cliente, hasta el despliegue y la configuración de un Web. Por ejemplo, el entorno de paginado de ASP.NET le permite construir interfases de usuario que separan limpiamente la lógica de su aplicación del código de su presentación, y maneja eventos con un modelo sencillo de procesamiento de formularios al estilo de Visual Basic. Adicionalmente, el CLR simplifica el desarrollo con servicios de código gerenciado, como el conteo automático de referencias y la limpieza automática de la memoria utilizada por su aplicación.

Gerenciabilidad

ASP.NET usa un sistema jerárquico de configuración, basado en archivos de texto, que simplifica la aplicación de parámetros de configuración al servidor y sus aplicaciones. Porque la información de configuración es almacenada como texto, nuevos parámetros pueden ser configurados sin recurrir a herramientas de administración locales. Esta filosofía de "cero administración local" también se extiende al despliegue de aplicaciones de ASP.NET. Una aplicación de ASP.NET se despliega a un servidor simplemente copiando los archivos necesarios al servidor. No hay que reiniciar el servidor, ni siquiera para reemplazar código compilado que ya está en servicio.

Escalabilidad y Disponibilidad

ASP.NET ha sido diseñado para la escalabilidad con características específicamente dirigidas a mejorar el funcionamiento de servidores racimados (clustered) y de servidores con procesadores múltiples. Los procesos del servidor son vigilados y gerenciados por el entorno del ambiente de ejecución de ASP.NET, así que si algún proceso se entorpece o se detiene, un nuevo proceso puede ser creado para reemplazarlo, lo cual ayuda a mantener la disponibilidad de su aplicación para manejar solicitudes de servicio.

Personalización y Extensibilidad

ASP.NET entrega una arquitectura bien formada que permite que los desarrolladores "enchufen" su código al nivel apropiado. De hecho, es posible el extender o reemplazar cualquier sub-componente del ambiente de ejecución de ASP.NET con un componente personalizado. La implementación de autenticación personalizada o de servicios de mantenimiento de estado nunca ha sido tan sencillo.

Seguridad

Con autenticación nativa de Windows y configuración individual por aplicación, usted puede estar tranquilo: sus aplicaciones están seguras.

6 SERVICIOS EN .NET.

Los servicios de Microsoft Windows, antes conocidos como servicios NT, permiten crear aplicaciones ejecutables de ejecución larga que se ejecutan en sus propias sesiones de Windows. Estos servicios pueden iniciarse automáticamente cuando se inicia el sistema, se pueden pausar y reiniciar, y no muestran ninguna interfaz de usuario. Esto hace que los servicios resulten perfectos para ejecutarse en un servidor o allí donde se necesite una funcionalidad de ejecución larga que no interfiera con los demás usuarios que trabajen en el mismo equipo. También puede ejecutar servicios en el contexto de seguridad de una cuenta de usuario específica, diferente de la del usuario que inició la sesión o de la cuenta predeterminada del equipo.

Puede crear servicios fácilmente mediante la creación de aplicaciones que se instalen como servicios. Por ejemplo, suponga que desea supervisar los datos del contador de rendimiento y reaccionar a valores umbral. Podría escribir una aplicación de servicios de Windows que lea los datos del contador de rendimiento, implementar la aplicación y comenzar a recoger y analizar los datos.

6.1 Aplicaciones de servicios y otras aplicaciones de Visual Studio .NET

Las aplicaciones de servicios funcionan, en varios aspectos, de forma diferente a muchos otros tipos de proyectos:

- El archivo ejecutable compilado que crea un proyecto de aplicación de servicios debe instalarse en el servidor para que el proyecto pueda funcionar de forma significativa. No es posible depurar o ejecutar una aplicación de servicios con las teclas F5 o F11; no es posible ejecutar inmediatamente un servicio o ejecutar su código paso a paso. En su lugar, es necesario instalar e iniciar el servicio y, a continuación, adjuntar un depurador al proceso del servicio.
- A diferencia de algunos tipos de proyectos, deberá crear componentes de instalación para las aplicaciones de servicios. Los componentes de instalación instalan y registran el servicio en el servidor y crean una entrada para el servicio con el Administrador de control de servicios de Windows.
- El método Main de la aplicación de servicios debe emitir el comando Run para los servicios que contiene el proyecto. El método Run carga los servicios en el Administrador de control de servicios del servidor adecuado. Si utiliza la plantilla de proyecto Servicios de Windows, este método se escribirá automáticamente. Tenga en cuenta que cargar un servicio no es lo mismo que iniciarlo.
- Las aplicaciones de servicios de Windows se ejecutan en un equipo Windows diferente al de las demás aplicaciones. Por este motivo, los cuadros de diálogo que produce una aplicación de servicios de Windows no se ven y pueden provocar que el programa deje de responder. De forma parecida, es recomendable registrar los mensajes de error en el registro de eventos de Windows, en lugar de hacerlo en la interfaz del usuario.
- Las aplicaciones de servicios de Windows se ejecutan en su propio contexto de seguridad y se inician antes de que el usuario inicie la sesión en el equipo Windows en el que se encuentran instaladas. Debe considerar detenidamente en qué cuenta de usuario se ejecutará el servicio; un servicio que se ejecute bajo la cuenta del sistema tendrá más permisos y privilegios que una cuenta de usuario.

6.2 Ciclo de vida de los servicios

Un servicio pasa por varios estados internos a lo largo de su vida útil. En primer lugar, se instala el servicio en el sistema en el que se ejecutará. Este proceso ejecuta los instaladores para el proyecto del servicio y carga el servicio en el Administrador de control de servicios del equipo. El Administrador de control de servicios es la utilidad central que proporciona Windows para administrar servicios.

Una vez cargado el servicio, es necesario iniciarlo. Al iniciar el servicio, se permite que empiece a funcionar. Puede iniciar un servicio desde el Administrador de control de servicios, desde el Explorador de servicios o desde el código mediante una llamada al método `ServiceController.Start`. El método **Start** pasa el procesamiento al método `OnStart` de la aplicación y procesa el código que allí se defina.

Un servicio en ejecución puede permanecer indefinidamente en este estado, hasta que se detiene o se pausa, o hasta que se apaga el equipo. Un servicio puede estar en uno de estos tres estados básicos: `Running`, `Paused` o `Stopped`. El servicio puede informar también del estado de un comando pendiente: `ContinuePending`, `PausePending`, `StartPending` o `StopPending`. Estos estados indican que se emitió un comando, por ejemplo, para hacer una pausa en un servicio en ejecución, pero que el comando aún no se ejecutó. Puede consultar la propiedad `ServiceController.Status` para determinar en qué estado se encuentra un servicio o utilizar el método `ServiceController.WaitForStatus` para ejecutar una acción cuando se produce cualquiera de estos estados.

Puede pausar, detener o reanudar un servicio desde el Administrador de control de servicios, desde el Explorador de servidores o por medio de llamadas a métodos en el código. Cada una de estas acciones puede llamar a un procedimiento asociado del servicio (`OnStop`, `OnPause` u `OnContinue`), en el cual es posible definir procesos adicionales que se ejecutarán cuando cambie el estado del servicio.

6.3 Tipos de servicios

Se pueden crear dos tipos de servicios en Visual Studio .NET. Los servicios que son el único servicio de un proceso se asignan al tipo **Win32OwnProcess**. Los servicios que comparten un proceso con otro servicio se asignan al tipo **Win32ShareProcess**. Para recuperar el tipo de servicio, consulte la propiedad `ServiceController.ServiceType`.

En ocasiones, es posible que vea otros tipos de servicios si consulta servicios existentes que no se hayan creado en Visual Studio .NET.

6.4 Los servicios y el componente ServiceController

El componente **ServiceController** se utiliza para conectar con un servicio instalado y manipular su estado. Al utilizar un componente **ServiceController**, se puede iniciar y detener un servicio, hacer una pausa y continuar con su funcionamiento, y enviar comandos personalizados a un servicio. Sin embargo, cuando cree una aplicación de servicio, no es necesario utilizar un componente **ServiceController**. En realidad, en la mayoría de los casos el componente **ServiceController** deberá existir en una aplicación separada de la aplicación de servicios de Windows que define el servicio.

6.5 Implementar e instalar servicios

Visual Studio .NET incluye componentes de instalación que pueden instalar recursos asociados a las aplicaciones de servicios. Los componentes de instalación registran un servicio individual en el sistema en el que se está instalando y permiten que el Administrador de control de servicios conozca la existencia del servicio.

Después de agregar instaladores a la aplicación, el siguiente paso consiste en crear un proyecto de instalación que instale los archivos de proyecto compilados y ejecute los instaladores necesarios para instalar el servicio. Para crear un proyecto de instalación completo, se debe agregar el resultado del proyecto de servicio al proyecto de instalación y, a continuación, agregar una acción personalizada para instalar el servicio.

Requisitos

- Los servicios deben crearse en un proyecto de aplicación de servicios de Windows o en otro proyecto compatible con .NET que cree un archivo .exe al ser generado y que herede de la clase **ServiceBase**.
- Los proyectos que contienen servicios de Windows deben tener componentes de instalación para el proyecto y sus servicios. Esto se puede hacer fácilmente desde la ventana Propiedades.

Namespace System.ServiceProcess

El espacio de nombres **System.ServiceProcess** proporciona clases que permiten implementar, instalar y controlar las aplicaciones desarrolladas como servicios de Windows. Los servicios son aplicaciones ejecutables de larga duración que se ejecutan sin interfaz de usuario. La implementación de un servicio implica la herencia de la clase **ServiceBase** y la definición de un comportamiento específico que se ha de procesar al pasar comandos de inicio, detención, pausa y continuación, así como el comportamiento personalizado y las acciones que se deben adoptar cuando se cierra el sistema. Parte de la definición de un servicio, es establecer sus condiciones de inicio, sus parámetros de ejecución, etc., ya que no cuentan con el control de un usuario.

Los servicios se instalan mediante una herramienta de instalación como InstallUtil.exe. El espacio de nombres **System.ServiceProcess** proporciona clases de instalación que escriben información sobre los servicios en el Registro. La clase **ServiceProcessInstaller** proporciona una clase englobadora que instala los componentes comunes a todos los servicios de una instalación. Para cada servicio se crea una instancia de la clase **ServiceInstaller** para instalar la funcionalidad específica de ese servicio.

La clase **ServiceController** permite conectarse a un servicio existente y manipularlo u obtener información sobre el mismo. Esta clase suele utilizarse en tareas administrativas, y permite iniciar, detener, pausar, continuar o ejecutar comandos personalizados en un servicio. Mientras que la clase **ServiceBase** define el procesamiento que realiza un servicio cuando se ejecuta un comando, **ServiceController** es el agente que permite llamar a esos comandos en el servicio.

Clases

Clase	Descripción
ServiceBase	Proporciona una clase base para un servicio que existirá como parte de una aplicación de servicio. ServiceBase deberá derivarse cuando se cree una nueva clase de servicio, como lo hacemos en nuestra implementación de nuestro servicio de atención de Message Queuing.
ServiceController	Representa a un servicio de Windows y permite conectarse a un servicio en ejecución o detenido, manipularlo u obtener información acerca del mismo.
ServiceControllerPermission	Permite el control de los permisos de seguridad de acceso al código para los controladores de servicios.
ServiceControllerPermissionAttribute	Permite comprobaciones declarativas de permiso de controlador de servicios.
ServiceControllerPermissionEntry	Define la unidad más pequeña de un permiso de seguridad de acceso a código establecida para ServiceController .
ServiceControllerPermissionEntryCollection	Contiene una colección con establecimiento inflexible de tipos de objetos ServiceControllerPermissionEntry .
ServiceInstaller	Instala una clase que extiende ServiceBase para implementar un servicio. La utilidad de instalación

	llama a esta clase al instalar una aplicación de servicio.
ServiceProcessDescriptionAttribute	Especifica una descripción para una propiedad o para un Evento.
ServiceProcessInstaller	Instala un ejecutable que contiene clases que extienden <u>ServiceBase</u> . Las utilidades de instalación, como InstallUtil.exe, llaman a esta clase al instalar una aplicación de servicio.
TimeoutException	Excepción que se inicia cuando ha transcurrido el tiempo de espera especificado.

Clase ServiceBase:

Proporciona una clase base para un servicio que existirá como parte de una aplicación de servicio. **ServiceBase** deberá derivarse cuando se cree una nueva clase de servicio. Derivando **ServiceBase** al definir la clase de servicio de una aplicación de servicio, definimos cualquier servicio útil al reemplazar los métodos OnStart y OnStop. Para obtener funcionalidad adicional, se puede reemplazar OnPause y OnContinue por un comportamiento específico en respuesta a cambios en el estado del servicio.

Un servicio es un ejecutable de ejecución larga que no admite una interfaz de usuario y que puede no ejecutarse en la cuenta de usuario que ha iniciado la sesión. El servicio puede ejecutarse sin que ningún usuario haya iniciado una sesión en el equipo.

De forma predeterminada, los servicios se ejecutan en la cuenta de sistema, que no es igual que la cuenta de administrador. De modo alternativo, puede utilizar ServiceProcessInstaller para especificar una cuenta de usuario en la que se ejecutará el servicio.

Un ejecutable puede contener más de un servicio, pero tiene que contener un componente ServiceInstaller independiente para cada uno de los servicios. La instancia de **ServiceInstaller** registra el servicio en el sistema. Asimismo, el instalador asocia cada servicio a un registro de eventos que puede utilizar para registrar comandos de servicio. La función main() del ejecutable define qué servicios deben ejecutarse. El directorio de trabajo actual del servicio es el directorio del sistema, no el directorio en el que está situado el ejecutable.

Cuando se inicia un servicio, el sistema busca el ejecutable y ejecuta el método **OnStart** de ese servicio, contenido dentro del ejecutable. Sin embargo, ejecutar el servicio no equivale a ejecutar el ejecutable. El ejecutable solamente carga el servicio. Se obtiene acceso al servicio (por ejemplo, lo inicia y lo detiene) mediante el Administrador de control de servicios.

El ejecutable llama al constructor de la clase derivada de **ServiceBase** la primera vez que se llama a Start para el servicio. Se llama al método de control de comandos **OnStart** inmediatamente después de que se ejecute el constructor. El constructor no se vuelve a ejecutar después de la primera vez que se ha cargado el servicio, por lo que es necesario separar el procesamiento realizado por el constructor del realizado por **OnStart**. Cualquier recurso que **OnStop** pueda liberar tiene que crearse en **OnStart**. La creación de recursos en el constructor impide que estos se creen adecuadamente si se inicia de nuevo el servicio una vez que **OnStop** ha liberado los recursos.

El Administrador de control de servicios (SCM, Service Control Manager) permite la interacción con el servicio. Se puede utilizar el SCM para pasar comandos Iniciar, Detener, Pausar, Continuar o comandos personalizados al servicio. El SCM utiliza los valores de las propiedades CanStop y CanPauseAndContinue para determinar si el servicio acepta los comandos Detener, Pausar o Continuar. Detener, Pausar y Continuar sólo estarán habilitados en los menús contextuales del SCM si la propiedad **CanStop** o **CanPauseAndContinue** correspondiente es **true** en la clase de servicio. Si está habilitado, el comando se pasa al servicio y se llama a **OnStop**, **OnPause** u **OnContinue**. Si la propiedad **CanStop**, CanShutdown o **CanPauseAndContinue** es **false**, no se procesará el método de control de comandos correspondiente (como **OnStop**), aunque se haya implementado el método.

Puede utilizar la clase **ServiceController** para realizar mediante programación lo que el SCM realiza mediante una interfaz de usuario. Es posible automatizar las tareas disponibles en la consola. Si la propiedad **CanStop**, **CanShutdown** o **CanPauseAndContinue** es **true**, pero no implementó el método de control de comandos correspondiente (como **OnStop**), el sistema inicia una excepción y omite el comando.

No es necesario implementar el método **OnStart**, **OnStop** ni ningún otro método en **ServiceBase**. Sin embargo, el comportamiento del servicio se describe en **OnStart**, por lo que, como mínimo, tendrá que reemplazarse este miembro. Se debe establecer el nombre de servicio en la función **main()** del ejecutable. El nombre de servicio que se establece en **main()** debe coincidir exactamente con la propiedad **ServiceName** del instalador del servicio.

Clase **ServiceController**

Se puede utilizar la clase **ServiceController** para conectarse y controlar el comportamiento de los servicios existentes. Al crear una instancia de la clase **ServiceController**, se pueden establecer las propiedades correspondientes de forma que interactúen con un servicio de Windows específico. De esta manera, se puede utilizar la clase para iniciar, detener o manipular el servicio.

Después de crear una instancia de **ServiceController**, se deben establecer dos propiedades en ella para identificar el servicio con el que interactúa: el nombre del equipo y el nombre del servicio que se desea controlar.

Normalmente, el autor del servicio escribe el código que personaliza la acción asociada a un comando específico. Por ejemplo, un servicio puede contener código que responda a un comando **ServiceBase.OnPause**. En tal caso, el procesamiento personalizado de la tarea **Pause** se ejecuta antes de que el sistema pause el servicio.

El conjunto de comandos que un servicio puede procesar depende de sus propiedades, por ejemplo, se puede establecer la propiedad **CanStop** en **false** para un servicio. Este valor procesa el comando **Stop** no disponible en ese servicio concreto; impide que se detenga el servicio del SCM al deshabilitar el botón necesario. Si se intenta detener el servicio desde el código, el sistema provoca un error y muestra el mensaje de error "Error al detener *servicename*".

Clase **ServiceInstaller**

ServiceInstaller no funciona de forma específica respecto al servicio al que está asociado. La utilidad de instalación lo utiliza para escribir valores del Registro asociados al servicio en una subclave dentro de la clave de Registro **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services**. El servicio se identifica mediante su propiedad **ServiceName** dentro de esta subclave. La subclave incluye también el nombre del ejecutable o archivo .dll al que pertenece el servicio.

Para instalar un servicio, se debe crear una clase de instalador del proyecto que herede de la clase **Installer**, y establecer el atributo **RunInstallerAttribute** de la clase en **true**. En el proyecto, deberíamos crear una instancia de **ServiceProcessInstaller** por cada aplicación de servicio y una instancia de **ServiceInstaller** para cada servicio de la aplicación. Finalmente, se agrega la instancia de **ServiceProcessInstaller** y las instancias de **ServiceInstaller** a la clase de instalador del proyecto.

Cuando se llama a la utilidad de instalación, ésta busca el atributo **RunInstallerAttribute**. Si el valor del atributo es **true**, la utilidad instala todos los servicios que se agregaron a la colección **Installers** asociados al instalador del proyecto. Si **RunInstallerAttribute** es **false** o no existe, la utilidad de instalación omite el instalador del proyecto.

Se puede modificar otras propiedades de **ServiceInstaller** antes o después de agregarlo a la colección **Installers** del instalador del proyecto. Por ejemplo, se puede establecer la propiedad **StartType** de un servicio de manera que inicie el servicio automáticamente al reiniciar o que, por el contrario, requiera que un usuario inicie el servicio de forma manual.

7 REFERENCIAS DE INTERÉS

- www.msdn.microsoft.com/library/default.asp – 20/5/2004
 \Network and Directory Services\Network Communications\MessageQueue
 \Ado.Net
 \Asp.Net
- Lenguaje de Programación C# - José Antonio González
- www.c-sharpcorner.com – 20/5/2004
- www.charphelp.com – 20/5/2004
- www.msdn.microsoft.com/vcsharp – 20/5/2004
- www.gotdotnet.com – 20/5/2004
- www.msdn.microsoft.com/netframework – 20/5/2004
- www.asp.net – 20/5/2004
- www.codeproject.com – 20/5/2004

PROYECTO DE GRADO – SISTEMAS DE WORKFLOW.

Javier Apaulaza, José Casarini, Rodrigo Recoba

Tutor: Guillermo Calderón
Usuario Responsable: Daniel Gómez

Instituto de Computación
Facultad de Ingeniería
Universidad de la República Oriental del Uruguay.

Marzo 2004

Resumen.

En este apéndice se presentan los diferentes componentes diseñados e implementados para la realización del prototipo, en forma más detallada que la realizada en el informe principal. Mediante este apéndice se pretende dar mayor profundidad a los componentes implementados, de forma de clarificar los diferentes módulos del sistema, y su utilidad dentro del sistema de workflow.

ÍNDICE

1	HERRAMIENTA DE DISEÑO DE PROCESOS.....	3
1.1	Introducción.....	3
1.2	Decisiones de diseño.....	4
1.3	Estudio de Herramientas Existentes.....	4
1.4	Herramienta Utilizada.....	4
1.5	Componentes involucrados en el diseño de procesos.....	5
1.6	Chequeos y restricciones sobre el flujo.....	7
1.7	Alcance.....	7
1.8	Primer Prototipo.....	8
1.9	Segundo prototipo.....	11
1.10	Conclusiones:.....	16
1.11	Métodos de Modelado de Flujos de Trabajo.....	17
2	ADMINISTRACIÓN DE LA INFORMACIÓN.....	22
2.1	XML – Esquemas XSD – Esquemas Relacionales -.....	22
2.2	Definición de esquemas.....	28
2.3	Requerimientos:.....	29
2.4	Diseño.....	30
2.5	Lógica de la Solución.....	33
2.6	Editor de Formularios.....	39
2.7	Generación del Formulario.....	46
2.8	Cargar Datos en el Formulario.....	49
2.9	Impactador en Base de Datos.....	57
3	COMUNICACIÓN CLIENTE – SERVIDOR.....	60
3.1	ICWorkflowClient.....	62
3.2	ICMessageService.....	66
3.3	ICWorkflow Dispatcher.....	73
4	MOTOR DEL SISTEMA DE WORKFLOW.....	76
5	MANEJADOR DE LISTA DE TAREAS.....	89
6	TOOL AGENTS.....	92
7	COMPONENTES REUSABLES.....	96
7.1	Gestor genérico de participantes.....	97
7.2	Manejador de sesiones.....	101
7.3	Generador y Evaluador de expresiones.....	103
7.4	Manejador de Base de Datos.....	105
7.5	Lenguaje para referenciar información del proceso.....	108
8	DISEÑO DE CLASES, PAQUETES, DEPLOY.....	111
9	DISEÑO DE BASE DE DATOS.....	112
10	Referencias De Interes:.....	125

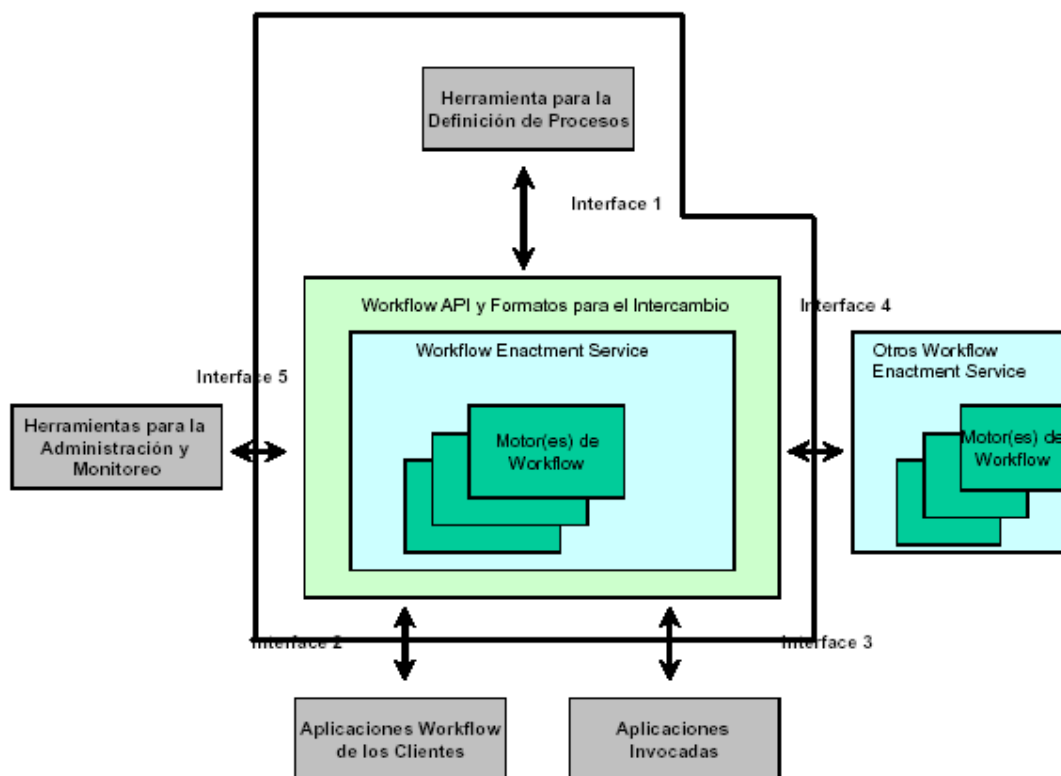
1 HERRAMIENTA DE DISEÑO DE PROCESOS.

1.1 Introducción

Como ya es sabido, el Workflow se encarga de automatizar procesos de negocio, los cuales deben ser definidos de alguna forma por los usuarios del mismo. Es por esto que se debe brindar una herramienta amigable que permita definir procesos.

Dicha herramienta, no solo permitirá diseñar procesos, sino que además será la encargada de traducir dicho proceso a un lenguaje que pueda ser interpretado por el motor del Workflow. Es en este punto en el que se debe tomar una importante decisión, ya que esta traducción puede realizarse de muchas formas.

En primera instancia podemos definir un lenguaje y realizar un motor que sepa interpretar al mismo. Esta opción parece ser bastante natural pero contiene una gran desventaja: solo podemos ejecutar flujos diseñados por nuestro editor. Es por esto que surge una idea más interesante y es la establecida por la WfMC, estandarizar el lenguaje de definición de procesos., de esta forma uno podría ejecutar procesos diseñados con otras herramientas e inclusive adquirir herramientas de diseño de procesos, siempre y cuando nuestro motor sepa interpretar dicho lenguaje.



La WfMC define entonces el XPD L como lenguaje de definición de procesos. Si bien las intenciones son buenas, en la práctica las cosas no funcionaron de la mejor manera, ya que no es el único estándar existente y no todos los Workflow lo implementan. De todas formas el estudio y comprensión de esta interfase nos ayudará a comprender mejor los componentes básicos de un flujo (Ver documento Interfases).

1.2 Decisiones de diseño.

A la hora de implementar la herramienta debemos considerar varios aspectos:

1. Estudio de herramientas existentes.
2. Herramienta utilizada.
3. Componentes involucrados en el diseño de procesos
4. Chequeos y restricciones sobre el diseño de procesos
5. Alcance.

1.3 Estudio de Herramientas Existentes.

Uno de los primeros puntos a atacar, es el estudio las herramientas existentes de definición de procesos, por dos motivos fundamentales: buscar una herramienta existente que se pueda utilizar sin necesidad de implementarla, aprender y observar los principales aspectos que esta debe incluir a la hora de implementar una.

Del estudio de las herramientas existentes, no encontramos ninguna que nos permitiera diseñar un proceso y la traduzca automáticamente a XPDL, si encontramos herramientas que nos permiten diseñar procesos fácilmente y además nos proveen la información necesaria para que nosotros interpretemos la misma y la traduzcamos a XPDL.

Es en este momento en el que surge la primera interrogante: ¿Utilizamos una de estas herramientas, o realizamos una desde el comienzo? Si bien contamos con herramientas que nos permiten diseñar fácilmente los flujos, estas tienen un costo por concepto de licencia que no siempre se esta dispuesto a gastar, es por este motivo que surge la posibilidad de desarrollar una herramienta desde cero, eliminado de esta forma estos gastos.

En nuestra situación la empresa INFOCORP no tenía inconveniente en afrontar dicho costo, creyendo conveniente la adquisición de algún producto como ser ya que los mismos presentan interfaces amigables para el usuario, siendo este uno de los factores más importantes en un producto de Workflow, de nada serviría si los usuarios no se sintieran cómodos o les fuera difícil utilizar el mismo. Además el costo de diseñar y desarrollar el mismo sería bastante elevado quitando tiempo y recursos al centro del problema. Por estos motivos se decidió realizar prototipos con diferentes productos que se detallarán en el siguiente punto.

1.4 Herramienta Utilizada.

A la hora de seleccionar herramientas para la realización de prototipos se buscaron aquellas que presentaran diferentes características. En primera instancia se estudió Microsoft Visio, estudiando la posibilidad de desarrollar un editor de flujos con la misma, y luego se estudio la posibilidad de realizarla utilizando Flowchart la cual nos permite diseñar los procesos mediante la Web.

El modo de uso de las mismas es bastante diferente, así como su interfaz. Microsoft Visio presenta un conjunto de librerías que permiten recorrer el diagrama, obtener los tipos de componentes del mismo y de esta forma ir generando el XPDL. Mientras que Flowchart genera un XML con la información del proceso, por lo cual la tarea es generar el XPDL a partir del XML.

1.5 Componentes involucrados en el diseño de procesos.

A la hora de diseñar un proceso debemos contar con diferentes componentes que nos permitan diseñarlos, los cuales enumeraremos a continuación:

- Inicio
- Fin
- Actividad
- Decisión
- Transition
- Join (And, Or)
- Split. (And, Or)

Esos elementos son fundamentales a la hora de diseñar un proceso, por lo cual deberán ser incluidos en nuestra herramienta de diseño de procesos. Los mismos serán explicados a continuación.

Inicio: Todo proceso debe tener un inicio por lo cual se precisa un componente con el cual se pueda indicar el inicio del proceso, el mismo podría permitir ingresar información extra como ser: el nombre del proceso, el creador, fecha de creación, etc.

Fin: Un proceso debe tener uno o más finales, siendo fundamental que cada posible flujo del mismo termine un nodo de fin.

Actividad: A medida que se va ejecutando el flujo se van realizando diferentes actividades, las cuales pueden ser de diferente índole, como ser: ejecutar una aplicación, enviar un mail, enviar una fax, etc. Por lo tanto necesitamos actividades que nos permitan indicar estas acciones. Podemos tener un componente diferente para cada tipo de actividad, o bien tener un componente genérico, que nos permita ingresar toda la información necesaria, opción que creemos más útil ya que de esta forma no es necesario modificar el Editor de flujos cada vez que se quiera agregar un tipo de actividad.

Decisión: También es necesaria la opción de decidir entre dos caminos según ciertos resultados o condiciones que se presenten. Es por esto que se incluye un componente que nos permite tomar una decisión según el cumplimiento o no de una determinada condición lógica.

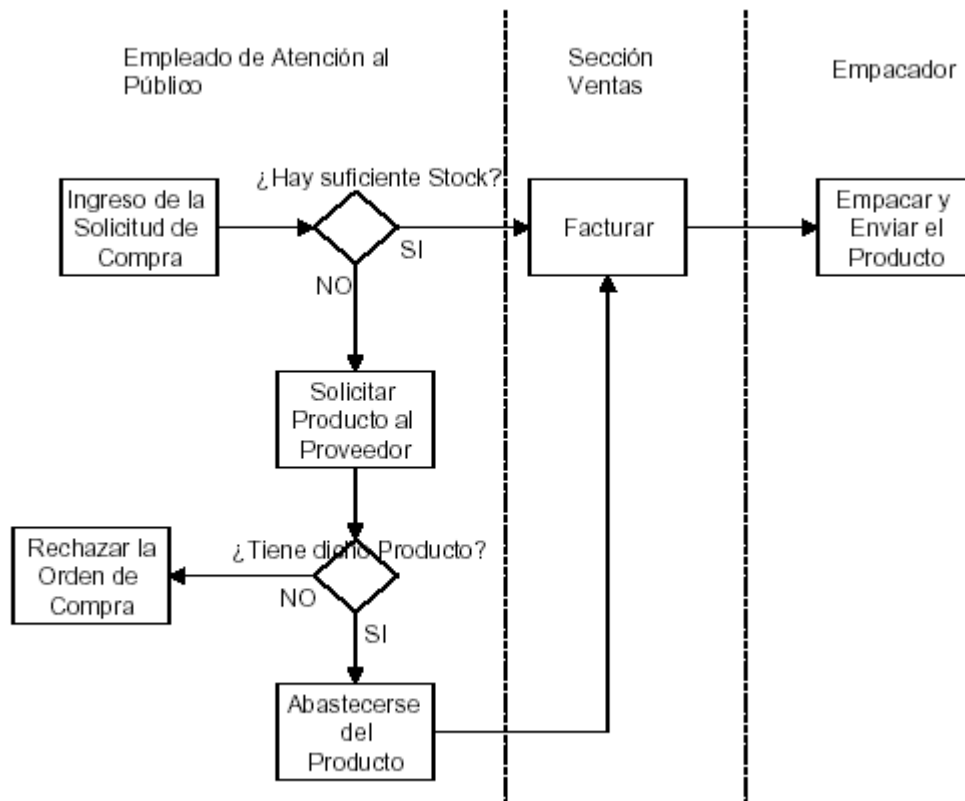
Transición: Las transiciones nos indican el sentido y orden en el que se recorre el flujo, son las responsables de guiarnos de un nodo a otro luego de finalizada una actividad, o tomada una decisión.

And Split: A partir de un lugar fuente, los documentos son distribuidos hacia varios destinos simultáneamente.

And Join: A partir de varios lugares fuentes, los documentos convergen, sincrónicamente, hacia un único destino.

Or Split: A partir de un lugar origen, los documentos toman un destino entre varios posibles.

Or Join: A partir de uno o más lugares de origen, dentro de varios posibles, convergen hacia un único destino (no se requiere sincronización).

Flujo de ejemplo.

1.6 Chequeos y restricciones sobre el flujo.

Es necesario realizar ciertos chequeos sobre el flujo diseñado para asegurar su correctitud. Entre ellos distinguimos.

- El proceso debe contener un único punto de partida.
- Un nodo de Inicio no puede tener transiciones entrantes
- Un nodo de Inicio debe tener una única transición saliente.
- Una Actividad puede tener una única transición entrante.
- Una Actividad puede tener una única transición saliente.
- Un nodo no puede tener una transición hacia si mismo.
- Un nodo de Decisión debe tener dos transiciones salientes.
- Un nodo de Fin no puede tener transiciones salientes.

1.7 Alcance

A la hora de desarrollar la herramienta consideraremos solamente alguno de los componentes detallados anteriormente. En primera instancia se implementaran todos los componentes exceptuando: Split y Join, por considerarlos de mayor complejidad y los mismos no aportarían nada a la hora de evaluar el prototipo. También se diseñaran diferentes tipos de actividades que permitan realizar determinados procesos, entre ellas destacamos:

- Invocación de scripts
- Invocación de llamadas a aplicaciones.
- Invocación de Web Services
- Participant Tasks (aquellas que son realizadas por algún recurso)

Se diseñará e implementará un sistema que permita gestionar participantes no solo se brindara la posibilidad de seleccionarlos de forma amigable, sino también crear grupos, relacionar participantes a grupos, modificarlos, etc.

También se implementaran todas y cada una de las restricciones mencionadas.

1.8 Primer Prototipo.

El primer prototipo fue realizado utilizando Microsoft Visio para permitir al usuario realizar los flujos en forma gráfica. Con tal motivo hubo que realizar un algoritmo que permita navegar el flujo y generar el XPDL asociado al mismo. Se tuvieron que realizar los diferentes "shapes", las restricciones, los formularios de ingreso de información, etc. Este prototipo cuenta con la desventaja de no ser Web pero si de uso práctico para usuarios. En la figura 2, se muestra un ejemplo diagrama realizado con Microsoft Visio.

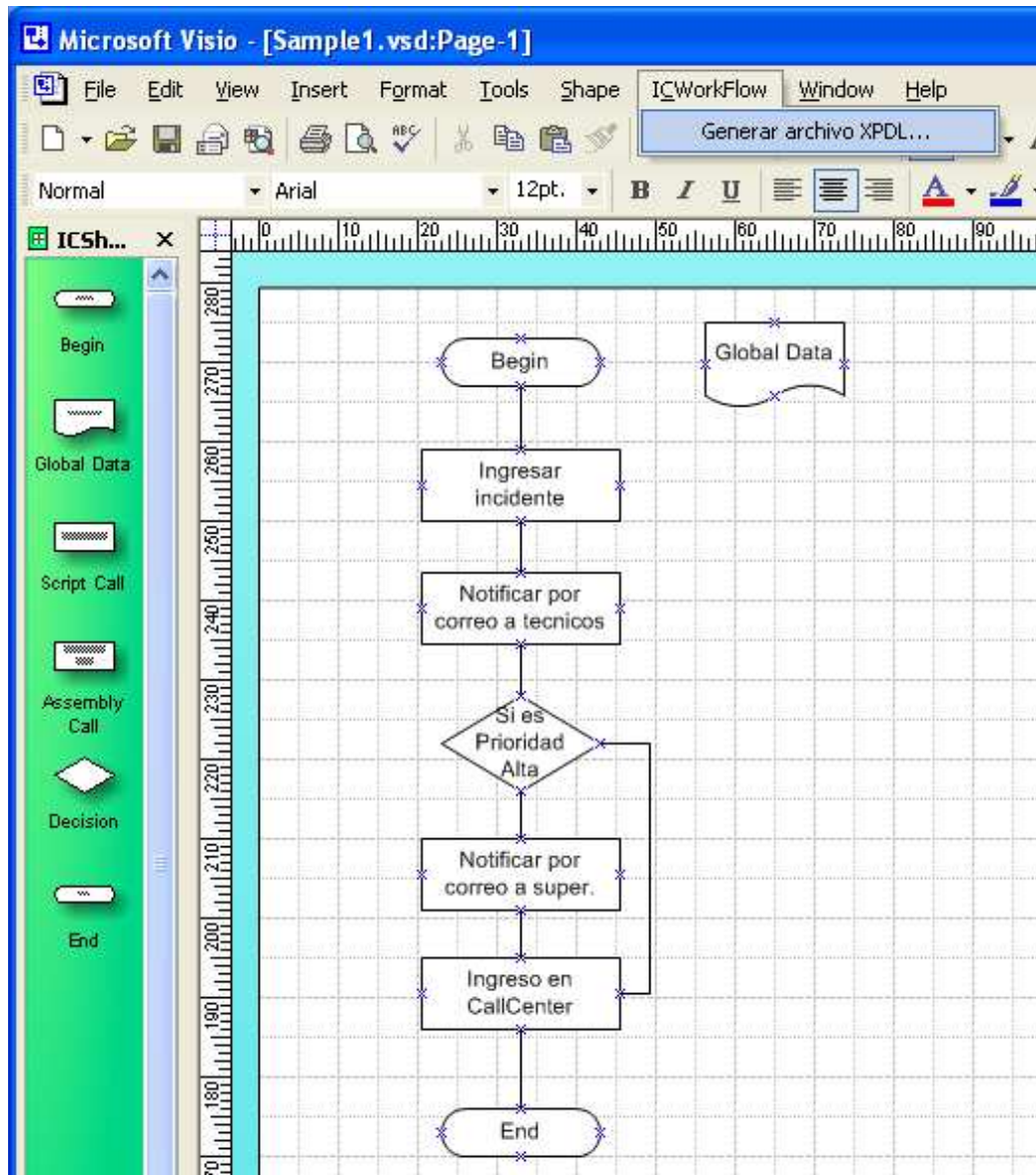


Figura 2

La realización de este prototipo fue bastante costosa, ya que hubo que estudiar las librerías ofrecidas por Visio, para trabajar con VB for Applications. Dicha librería es extensa y permite un gran abanico de posibilidades, agregándole complejidad al problema. Además hubo que estudiar como generar objetos COM, ya que todos los objetos especificados en XPDL, fueron realizados en C#. Por tal motivo, para poder usarlos e instanciarlos, tuvimos que generarlos como objetos COM.

Para aclarar un poco el funcionamiento de la herramienta, se explicará brevemente su funcionamiento. Como se puede visualizar en la Figura 2, la herramienta consiste en un conjunto de shapes que se encuentran a la izquierda, entre los que destacamos inicio, fin, decisión, actividad, global data, etc. Para realizar un flujo simplemente alcanza con ubicar las actividades que queramos sobre el área de diseño, con ciertas restricciones:

- Debe haber un nodo de inicio y uno de fin
- Inicio debe tener una transición de salida
- Fin debe tener una transición de entrante
- Las condiciones deben tener dos transiciones salientes
- Global data no puede tener transiciones entrantes ni salientes.

Al colocar un nodo en el área de trabajo, se desplegará un formulario que nos permitirá ingresar la información correspondiente como el que se muestra en la Figura 3, por ejemplo, al agregar un nodo de inicio, podremos ingresar el nombre del proceso; al agregar un nodo de global data, podremos ingresar las variables del sistema, etc.

Propiedades de actividad - Script Call

Workflow

Propiedades de actividad - Script Call

Descripción: Ingresar incidente

Tipo: VBScript

Codigo fuente:

```
'Activity - Script Call
' Main
'
Function Main (ProcessData, ActivityData)
'Put script code here
End Function
```

Verificar

Parametros:

Id	Nombre	Tipo	Valor
1	Debug	Texto	1

Agregar

Aceptar Cancelar Eliminar

Aceptar Cancelar

Figura 3

Mediante esta herramienta podemos realizar cualquier flujo que queramos, utilizando los nodos provistos, exceptuando aquellos flujos que involucren:

- Joins
- Fork

Si bien las actividades que asisten a la creación del flujo (ScriptCall, assemblyCall) son pocas, pueden agregarse con facilidad, permitiendo generar cualquier flujo que deseemos, respetando las descripciones anteriormente descritas y con las limitaciones mencionadas.

Como se puede apreciar, la interfaz no es muy atractiva para el usuario, pero si práctica y de uso sencillo. Una vez realizado el flujo se debe generar el XPDL utilizando el menú IC-Workflow, "Generar archivo XPDL", dicha opción invoca una macro realiza en VB. For Applications, que genera el XPDL asociado al flujo (ver Figura 4.)

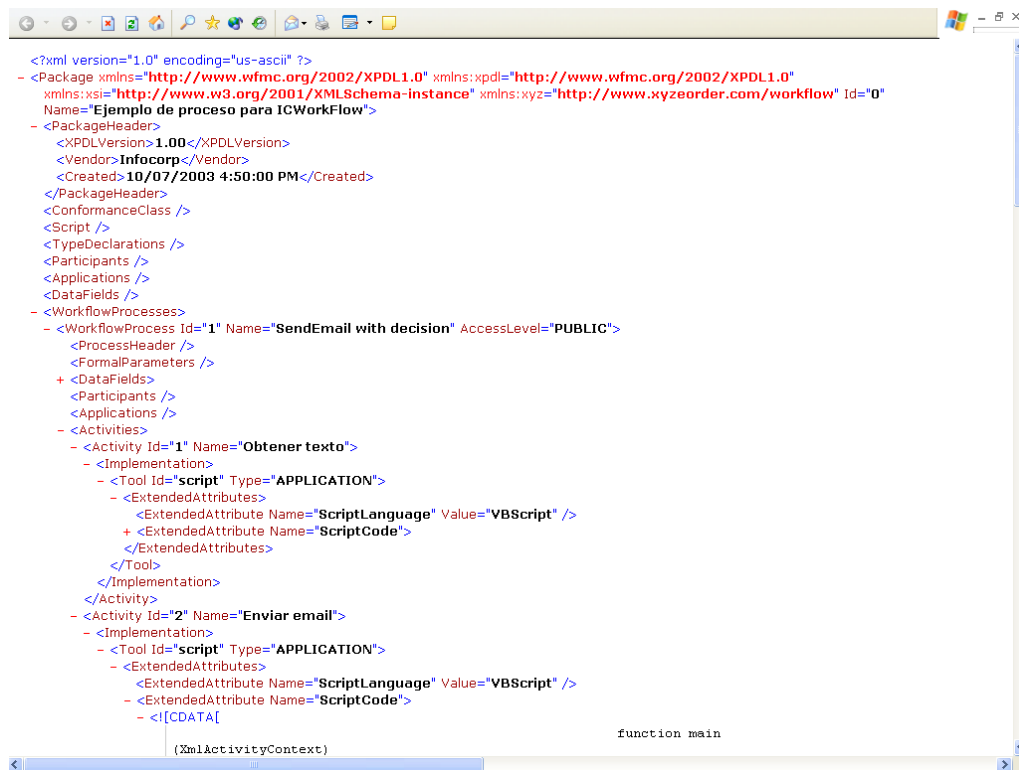


Figura 4

Para realizar el XPDL, se va navegando el flujo realizado, creando los objetos correspondientes, que son aquellos definidos mediante el esquema del XSD, dichos objetos fueron creados automáticamente utilizando .NET, y fueron generados en C#. Para poder accederlos en VB, hubo que generarlos como componentes COM.

El algoritmo que genera el XPDL, va navegando por el flujo, creando los objetos correspondientes y una vez terminado se serializa el objeto principal (Package), de esta forma ya estamos generando el XPDL.

Este prototipo no fue de total satisfacción, por la complejidad de su realización, por su amigabilidad hacia el usuario, y por no ser web como el resto del sistema. Por este motivo es que se hizo un segundo y definitivo prototipo que se describe a continuación.

1.9 Segundo prototipo.

El segundo prototipo fue realizado utilizando flowChart, una herramienta que permite realizar diagramas en forma web. Dicha herramienta es bastante atractiva (como se puede apreciar en la Figura 5), y de uso sencillo para el usuario y para el programador, ya que hay que definir los nodos a utilizar y la herramienta genera automáticamente un XML que describe dicho flujo.

Por esto, lo único que hay que realizar es definir los nodos a utilizar, implementar los formularios de ingreso de información, y realizar una transformación del XML a XPDL.

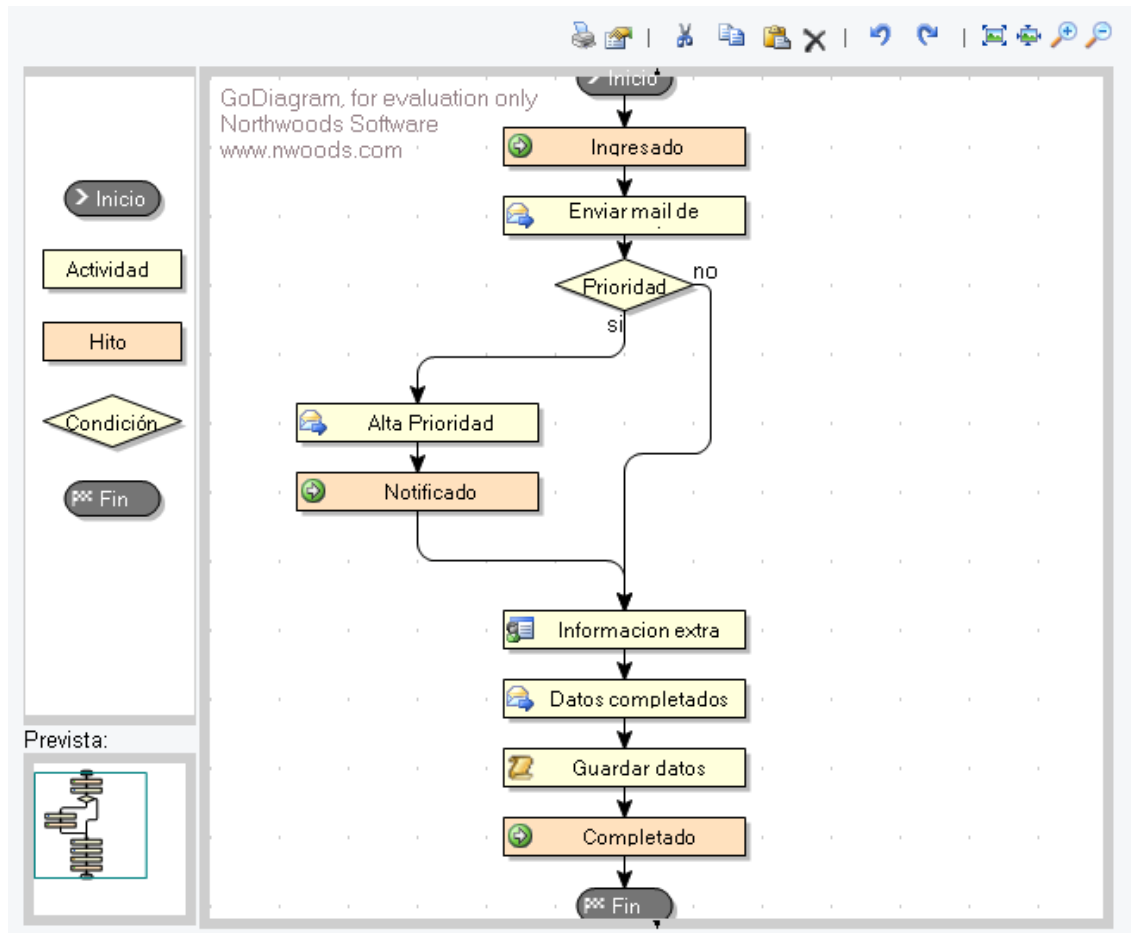


Figura 5

Al igual que en el primer prototipo realizado. Esta herramienta presenta una lista de nodos, presentes en la izquierda de la Figura 5, con los cuales podemos diagramar cualquier flujo exceptuando:

- Forks
- Joins.

A diferencia del primer prototipo, en este caso tenemos un solo nodo de actividad, el cual, al ser arrastrado sobre el área de diseño, nos permite seleccionar que tipo de actividad se desea como se muestra en la Figura 6, indicándonos su utilidad (Ver Figura 7). De esta forma, el agregar un nuevo tipo de actividad al sistema, no implica modificaciones en el editor de flujos, ya que los tipos disponibles están almacenados en una tabla. Una vez ingresado el nodo se presenta un formulario que permite ingresar la información, idéntico al realizado en el primer prototipo (Figura 8) La información agregada a cada actividad es almacenada y agregada en el XML de salida, por lo cual debe ser considerada en la transformación.

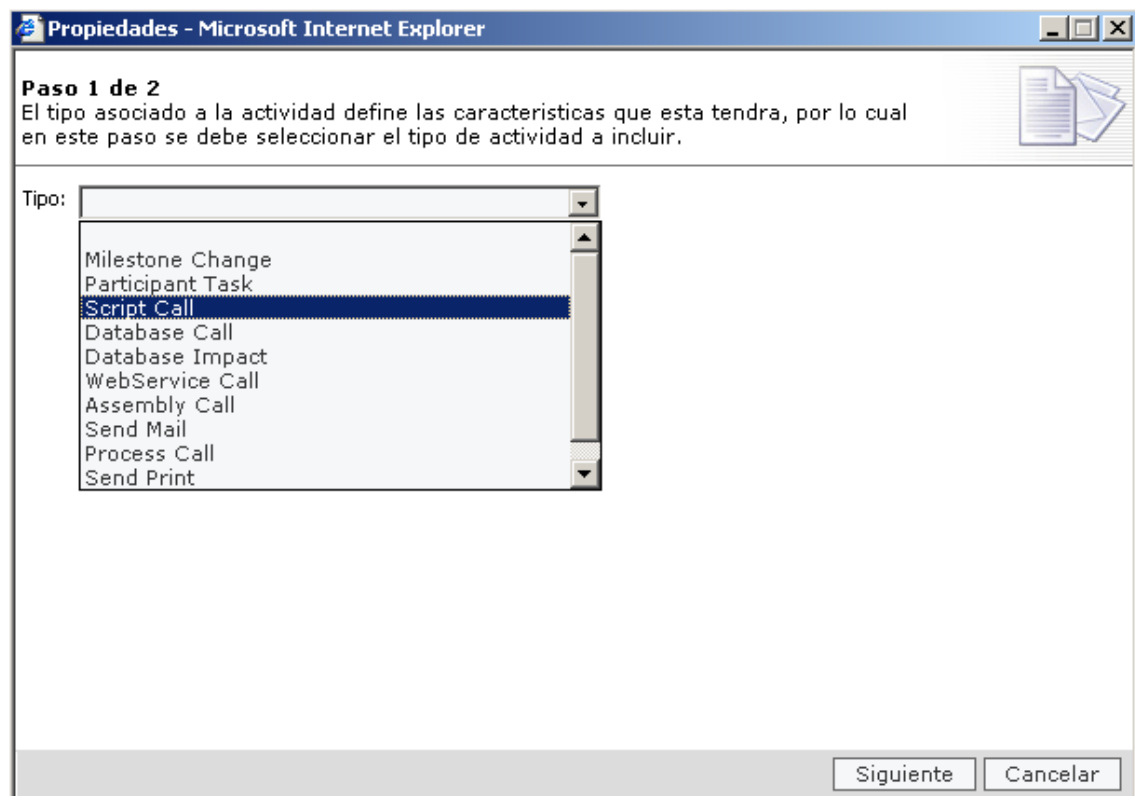


Figura 6

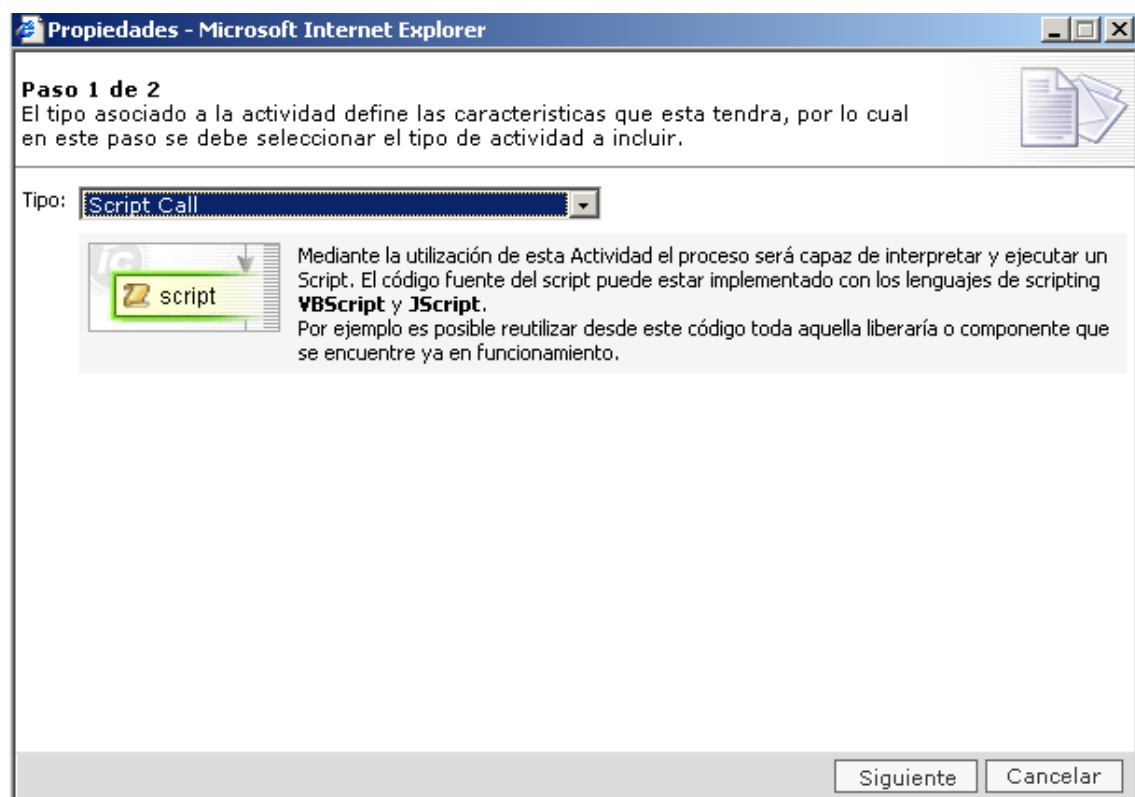


Figura 7

Propiedades
Ingrese aquí las propiedades asociadas al elemento seleccionado.

Título: Script de Prueba

Descripción: Este Script es una prueba

Tipo: VBScript

Código fuente:

```
'Activity - Script Call
'Main
Function Main (ProcessData, ActivityData)
'Put script code here
End Function
```

Verificar

Parámetros:

Id	Nombre	Tipo	Valor		
1	Parametro1	String	Rodrigo	Modificar	Eliminar

Aceptar Cancelar

Figura 8

En este prototipo no tenemos una actividad o nodo de tipo Global Data, ya que las variables globales se ingresan en el formulario de información del proceso (Del nodo inicio), junto con los demás datos del proceso de negocio, como ser, la información de entrada al proceso, los datos que las diferentes actividades modificarán en común y aquellas que se utilizan al tomar decisiones. Dicha información es ingresada al crear el proceso y en el nodo de inicio. En la Figura 9 se muestra el primer formulario que se genera al iniciar un proceso, este permite el ingreso de información necesario, según un esquema relacionado al flujo. Dicho esquema se genera al crear el proceso utilizando el Schema Editor.

Creación de instancia de proceso - Paso 2 de 2

Cancelar | < Anterior | Guardar y nuevo

Información general

Sistema: Servicios

Proceso: Reclamo Call Center

Datos de ingreso al proceso

IDReclamo: 1

Prioridad:

Fecha del reclamo: 10/5/2003

Nombre: Rodrigo

Apellido: Recoba

Email: rrecoba@hotmail.com

Reclamo: Servidor caído

Figura 9.

Además se agrego un nodo nombrado "HITO", que permite identificar con facilidad el estado en que se encuentra el proceso de negocio, esta actividad es la única reconocida por el sistema de Workflow, ya que dicha información es persistida y manejada por el mismo. El motivo es que es información específica del avance del proceso, que puede consultarse más frecuentemente y por ello se maneja de forma diferenciada.

Otro nodo importante del edito de flujos es el nodo de decisión, dicho nodo debe permitir ingresar una condición, la cual es evaluada por el motor para saber el camino a seguir. En la Figura 11, se indica un formulario típico de decisión.

Propiedades
Ingrese aqui las propiedades asociadas al elemento seleccionado.

Título:

Descripción:

Condición:

Id	Campo	Operador	Valor	Y/O		
1	{statedata.xmlinput.//Prioridad}	GREATER	5	AND	Modificar	Eliminar
2	{statedata.xmlinput.//Prioridad}	LESSEQUAL	10	AND	Modificar	Eliminar

Figura 11

Tanto al agregar condiciones de decisiones como al pasar parámetros a las aplicaciones externas, necesitamos, en la mayoría de las ocasiones, acceder a información que es ingresada en tiempo de ejecución del proceso, es decir, no tenemos disponible su valor en tiempo de ejecución.

Por ejemplo, como se muestra en la figura 11, una condición puede depender de la prioridad que se haya ingresado en la entrada al proceso. O bien, como se indica en la Figura 8, un parámetro que se le pasa a una aplicación externa puede, en lugar de ser una constante, un valor ingresado en cualquier etapa del proceso, ya sea en la entrada, en una actividad que haya modificado las variables globales, en información ingresada en una tarea de un participante.

Dicha información se encuentra de la siguiente forma:

1. GlobalData: Contiene las variables globales del proceso.
2. StateData: Contiene el XML de entrada al proceso y la información modificada por los participantes.

Para hacer referencia a una consulta sobre el xml de entrada, se indica de la siguiente forma:

- **Statedata.xmlInput.//consulta_XPath**

Mientras que para hacer referencia sobre una variable global, se utiliza el siguiente esquema:

- **GlobalData.VariableGlobal**

De esta forma, podemos ingresar condiciones según información ingresada en tiempo de ejecución la cual será sustituida por el motor de WorkFlow, el cual a la hora de evaluar una condición o pasar parámetros a una aplicación externa, reconocerá este tipo de expresiones, sustituyéndola por el valor que corresponda.

Una vez terminado el flujo, la herramienta brinda un xml que describe el flujo, con la información referente a cada nodo, por lo tanto, se implemento una funcionalidad que permite traducir dicho XML, al lenguaje de definición de procesos que interpreta el motor de WorkFlow (XPDL).

Dicha traducción va creando actividades según su tipo (información que es brindada en el XML, como se puede apreciar en la Figura 12), y cargando la información de ellas en los extended attributes, los cuales luego serán reconocidos por cada tipo de actividad en particular. También se van generando las transiciones con sus condiciones. Una vez terminada la transformación, se almacena la definición del proceso en un repositorio central.

```
- <flowchart version="1" name="IC-WorkFlow Flow">
  <node kind="1" id="0" pidbottom="1" text="Inicio" x="92.5" y="30"><shapedata><title>Solicitud de Vacaciones</title><description></description></node>
  <node kind="2" id="20" pidtop="21" text="Fin" x="212.5" y="630" />
  <node kind="5" id="24" pidtop="25" pidbottom="26" text="Solicitud Ingresada" x="50" y="69"><shapedata><idtoolagent>ICToolAgentMilestoneChange</idtoolagent><idtoolagentimage>floweditor_milestone.png</idtoolagentimage><description>La solicitud fue ingresada.</description><milestone>Ingresado, a Aprobar.</milestone></shapedata></node>
  <node kind="5" id="31" pidtop="32" pidbottom="33" text="Solicitud Aprobada" x="50" y="269"><shapedata><idtoolagent>ICToolAgentMilestoneChange</idtoolagent><idtoolagentimage>floweditor_milestone.png</idtoolagentimage><description>La solicitud fue Aprobada, debe ser Verificada</description><milestone>Aprobado, a Verificar.</milestone></shapedata></node>
  <node kind="5" id="40" pidtop="41" pidbottom="42" text="Solicitud Rechazada" x="250" y="469"><shapedata><idtoolagent>ICToolAgentMilestoneChange</idtoolagent><idtoolagentimage>floweditor_milestone.png</idtoolagentimage><description></description><milestone>Rechazada</milestone></shapedata></node>
  <node kind="5" id="49" pidtop="50" pidbottom="51" text="Solicitud Aprobada" x="50" y="469"><shapedata><idtoolagent>ICToolAgentMilestoneChange</idtoolagent><idtoolagentimage>floweditor_milestone.png</idtoolagentimage><description></description><milestone>Aprobada</milestone></shapedata></node>
  <node kind="3" id="46" pidtop="47" pidbottom="52" text="Notificacion" x="170" y="549"><shapedata><idtoolagent>ICToolAgentSendMail</idtoolagent><idtoolagentimage>floweditor_sendmail.gif</idtoolagentimage><description>Se notifica al usuario el resultado de su solicitud.</description><smtpserver>Apolo</smtpserver><from>IC-Admin@ICCorp.com</from><to></to><cc></cc><bcc></bcc><subject>Solicitud de Licencia</subject><message>Resultado Aprobacion: {StateData.XmlInput.//Decripcion2}</message></shapedata></node>
  <link id="6" from="4" to="8" />
  <link id="12" from="1" to="25" />
  <link id="17" from="26" to="3" />
  <link id="19" from="10" to="32" fromlabel="si" />
  <link id="23" from="33" to="14" />
  <link id="27" from="9" to="41" fromlabel="no" />
  <link id="30" from="15" to="37" />
</flowchart>
```

Figura 12

Cómo se visualiza en la Figura 12, el XML posee cada nodo y las transiciones entre ellos. Los nodos son identificados mediante el tag <Node>, mientras que las transiciones se identifican con el tag <link>.

Los nodos tienen los siguientes atributos

- **Kind:** indica el tipo de actividad de que se trata (tool, root, etc.).
- **PidBottom, pidTop:** representa los conectores, son dichos identificadores a los que hacen referencia los links para indicar destino y fin.
- **Text:** descripción o título de la actividad
- En su texto tiene un xml, con nodo principal shapeData. En éste, se mantiene toda la información referente a la actividad, por ejemplo, si es un toolAgent, el tipo de toolAgent, los parámetros que recibe, etc.; si es una condición, la expresión que representa la condición a evaluar.

Los links tienen los siguientes atributos:

- **From:** indica el nodo inicio de la transición, hace referencia a uno de los "pid" de una actividad.
- **To:** indica el nodo Fin de la transición, hace referencia a uno de los "pid" de una actividad.
- **FromLabel:** Indica la etiqueta de la transición.

Cómo se puede ver, una transición no tiene la condición por la cual es evaluada. Esto es porque la condición es agregada en una actividad de un determinado tipo, el cual el motor sabrá interpretar.

Por lo tanto, para generar el lenguaje de definición de procesos, alcanza con ir recorriendo los nodos y creando los objetos (XPDL) correspondientes, según su tipo (kind), e ir cargando su información. Esta información es la almacenada en ShapeData, y es cargada en los extended attributes si esta es información particular de la actividad en cuestión. Si dicha información es de relevancia para el motor, como podría ser, fecha de finalización, es cargada en los atributos ya definidos en XPDL y no en extended Attributes.

Luego se recorren las transiciones, señaladas como link, analizar los from / to que hacen referencia a los conectores de los nodos, y se van generando las transiciones, generando el XPDL que se visualiza en la Figura 13. Estas transiciones, en general, no tienen una condición asociada, y evalúan siempre en verdadero, exceptuando aquellas condiciones que parten de un nodo decisión, en este caso, la condición es la ingresada en el dicho nodo.

Cómo resultado del proceso anteriormente descrito, se obtiene una definición de proceso, según, la especificación comprendida por el motor de Workflow, como puede visualizarse en la Figura 4

1.10 Conclusiones:

El segundo prototipo fue el que más nos conformó por su atractivo y facilidad de uso, además es web y genérico, ya que no es necesario modificarlo para agregar nuevos tipos de actividades, alcanza con agregarlos en una tabla.

Fue de gran importancia realizar el lenguaje para acceder a información de proceso, para darle mayor potencial a la herramienta, ya sea para pasar dicha información a las aplicaciones como para evaluar condiciones. La evaluación de condiciones trae aparejada otra complejidad, ya que no alcanza con traducir la expresión, sino que también hay que evaluarla.

1.11 Métodos de Modelado de Flujos de Trabajo

A continuación se describirán las características principales de tres de las principales metodologías que permiten el modelado de flujos dentro de las organizaciones.

Modelo ICN (Information Control Net).

A continuación se presentará uno de los métodos usados para modelar procesos de flujos de trabajo denominado ICN o Information Control Net desarrollado por Xerox en Palo Alto [Ellis].

El modelo ICN permite representar procesos de trabajo de una empresa bajo la forma de un grafo de actividades. ICN visualiza a la organización como un conjunto de procedimientos interrelacionados. Cada uno de estos procedimientos se caracteriza por medio de un grafo que describe las actividades que se realizan durante el mismo.

Un procedimiento, es entonces, un conjunto de actividades ejecutadas automática o manualmente por usuarios, grupos de usuarios, computadores, etc. en un orden específico y relacionadas por restricciones de precedencia. Como por ejemplo, el proceso de una orden de compra.

Las actividades son unidades de trabajo que se realizan en forma manual, automática o ambas. Las mismas pueden ocurrir en paralelo durante un procedimiento, o puede existir una relación entre dos o más actividades.

La Figura 13 muestra como son realizadas las actividades 3 4 y 5 en forma paralela, después de haberse cumplido la actividad 2.

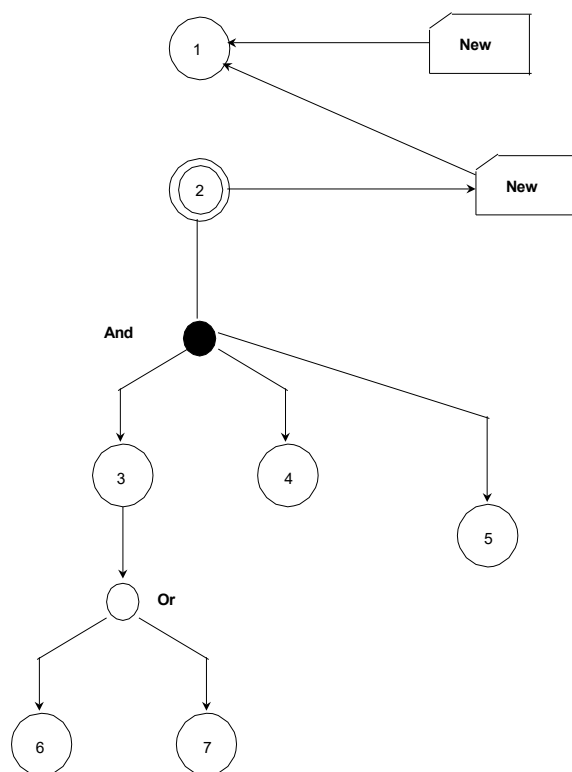


Figura 13
Grafo de Actividades ICN

Desde el punto de vista de la notación, las actividades son representadas como círculos simples en caso de que los responsables sean usuarios definidos o círculos dobles en caso de que los responsables sean grupos de usuarios o roles genéricos.

Los flujos de control se representan con arcos orientados que relacionan actividades (relaciones de precedencia). Los flujos de información se representan mediante arcos orientados que unen actividades con repositorios de información (archivos, documentos, mensajes). En este segundo caso, los flujos pueden ser de escritura o lectura. En otras palabras un arco entre dos actividades siempre representa un flujo de control. Los formularios o carpetas de trabajo se mueven entre actividades pasando por depósitos intermedios, secuencialmente o en paralelo.

Se definen dos tipos de puntos de sincronización entre actividades:

1 Manejo de actividades en paralelo

- ☐ **AND FUENTE** Bifurcación del flujo tal que las actividades que suceden un "And Fuente" son ejecutadas en paralelo.
- ☐ **AND SINCRONIA** Punto de sincronización del flujo; todas las actividades que preceden o convergen en un "And Sincronía" deben haber terminado para seguir con la actividad siguiente.

2 Manejo de actividades que dependen de una decisión

- ☐ **OR FUENTE** Bifurcación condicional tal que una o un subconjunto de las posibles actividades que suceden un "Or Fuente" van a ser ejecutadas.
- ☐ **OR SINCRONIA** Punto de sincronización a diferencia del AND la actividad que suceden un "Or Sincronía" es ejecutada si alguna de las tareas que la preceden terminó.

A cada "Or Fuente" debe corresponder un "Or Sincronía" para realizar la sincronización de las actividades dentro del grafo.

Redes de Petri.

En la sección anterior se describió la metodología ICN para la modelación de flujos de trabajo. En esta sección se presentará otra de las metodologías más usadas para el modelado de flujos de trabajo, conocidas como las redes de Petri [Peter].

Las redes de Petri representan modelos abstractos y formales de flujos de información. Buscando describir y analizar flujos de información y flujos de control en sistemas y en particular en aquellos sistemas que pueden exhibir actividades asincrónicas y concurrentes. El mayor uso de las redes de Petri se ha producido en el modelado de sistemas de eventos discretos, donde es posible que ciertos eventos ocurran simultáneamente, con la presencia de restricciones en la simultaneidad, precedencia o frecuencia de estas ocurrencias.

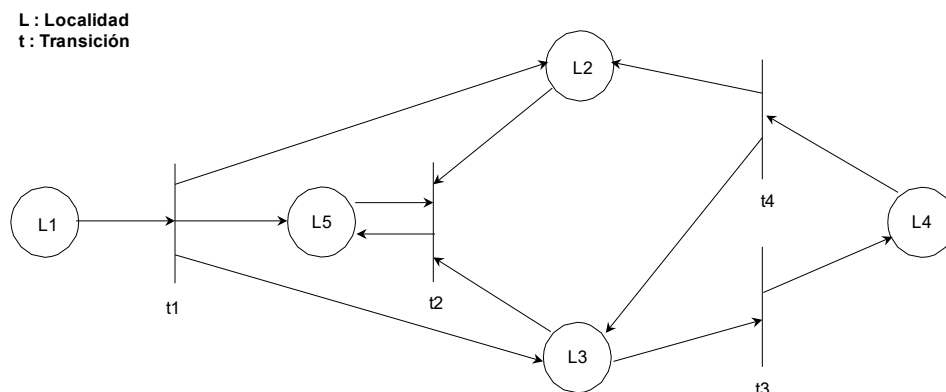


Figura 14
Red de Petri

La Figura 14 muestra una red de Petri simple. El grafo contiene dos tipos de nodos: círculos (localidades) y barras (transiciones). Estos nodos están conectados a través de arcos dirigidos, desde localidades a transiciones y desde transiciones a localidades. Si un arco está dirigido desde el nodo i al nodo j , entonces i es una "entrada" de j y j es una "salida" de i . La ejecución de una red de Petri está controlada por la posición y el movimiento de ciertas marcas llamadas "fichas" y representadas por puntos negros a través de la red. Las reglas son simples. Las fichas se mueven cuando se activan las transiciones de una red. Una transición debe estar habilitada para activarse. Una transición se habilita cuando todas sus entradas están marcadas con una ficha. La transición se activa quitando las marcas de los nodos de entrada y marcando los nodos de salida de la transición. La distribución de las fichas en una red de Petri marcada define el estado de la misma. El estado puede cambiar como resultado de la activación de transiciones.

Las redes de Petri se concentran en dos aspectos de los sistemas que pretenden modelar: los eventos y las condiciones imperantes (y también la relación entre ambos). En un sistema, en contexto de un determinado instante de tiempo, pueden existir determinadas condiciones. La existencia de estas condiciones puede causar la ocurrencia de ciertos eventos. Estos eventos pueden cambiar el estado de un sistema, eliminando algunas de las condiciones existentes y crear nuevas condiciones.

Las propiedades más destacadas de las redes de Petri se mencionan a continuación :

- Permiten modelar eventos concurrentes en paralelo
- Son de naturaleza asincrónica
- Su ejecución es no determinística

Hasta el momento se ha descrito a grandes rasgos lo que son las redes de Petri. En la siguiente sección se examinarán con más detalle cada uno de los elementos y la estructura general con la que se conforman las redes de Petri.

Estructura de las redes de Petri : modelo formal

Las redes de Petri están compuestas por dos tipos de elementos: un conjunto de localidades (L) y un conjunto de transiciones (T). Para completar la definición, debemos definir la relación entre localidades y transiciones. Esto puede realizarse especificando dos funciones que conectan transiciones con localidades: E la función de entrada y S, la función de salida. La función de entrada define para cada transición t_j , el conjunto de localidades de entrada para la transición $E(t_j)$. La función de salida S define, para cada transición t_j , el conjunto de localidades de salida para esa transición $S(t_j)$. Estos cuatro ítems definen la estructura de una red de Petri. Formalmente, una red de Petri C se define como :

$$C = (L, T, E, S)$$

Como ejemplo consideremos la estructura de red que sigue :

$$C = (L, T, E, S)$$

$$L = \{l_1, l_2, l_3, l_4, l_5\}$$

$$T = \{t_1, t_2, t_3, t_4\}$$

$$E(t_1) = \{l_1\}$$

$$E(t_2) = \{l_2, l_3, l_5\}$$

$$E(t_3) = \{l_3\}$$

$$E(t_4) = \{l_4\}$$

$$S(t_1) = \{l_2, l_3, l_5\}$$

$$S(t_2) = \{l_5\}$$

$$S(t_3) = \{l_4\}$$

$$S(t_4) = \{l_2, l_3\}$$

A pesar de ser útil, la definición anterior no es la más adecuada para ilustrar una serie de conceptos de una manera más informal e intuitiva. Para estos propósitos, se usa una representación distinta llamada grafo de una red de Petri.

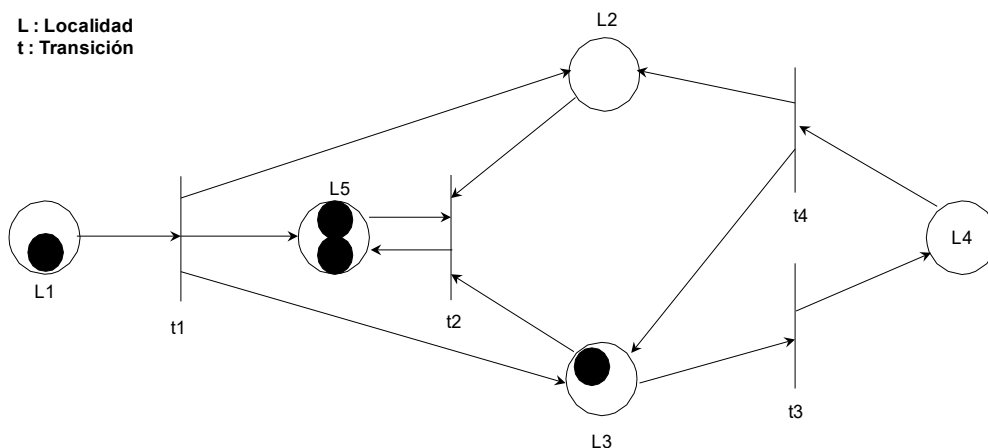


Figura 15
Red de Petri Marcada

Un marcado M de una red de Petri corresponde a una distribución de fichas en las localidades de la red. La cantidad y ubicación de las fichas en una red puede cambiar durante su ejecución. El vector $M = (m_1, m_2, m_3, \dots, m_n)$ indica, para cada localidad en la red de Petri, el número de fichas en esa localidad. Podemos también definir una función de marcado $M(l_j) = m_j$

La figura 15 es un ejemplo de una red de Petri marcada . Representa la estructura anterior con un marcado $M = (L, T, E, S, M)$

Reglas de ejecución

Una red de Petri se ejecuta activando transiciones. Una transición puede activarse si esta habilitada. Una transición está habilitada si cada una de sus localidades de entrada tiene al menos una ficha. En la figura 15 por ejemplo, la transición t_2 se habilita si l_2 tiene al menos una ficha, l_3 tiene al menos una ficha y l_5 tiene al menos una ficha. Una transición se activa eliminando una ficha de cada una de sus entradas y depositando una ficha en cada una de sus salidas. La activación de una transición típicamente cambiará el marcado de una red de Petri, M a un nuevo marcado M' .

Dada una red de Petri y un marcado inicial M_0 , podemos ejecutar la red mediante activaciones de transiciones sucesivas, hasta llegar a un marcado en el cual no existen transiciones habilitadas, en cuyo caso la ejecución de la red debe cesar. La secuencia de marcados y la secuencia de transición, proveen un registro de la ejecución de la red.

Extensiones y subclases

Ya que no siempre es fácil modelar ciertos eventos o condiciones, se han hecho importantes esfuerzos para extender el poder del modelado de las redes de Petri. Se han definido también subclases, restringiendo su estructura, buscándose mejorar con ello su potencial de análisis. Destacan dos de estas subclases: las máquinas de estado y los grafos marcados.

Un grafo marcado es una red de Petri en que cada localidad tiene exactamente una transición de entrada y una transición de salida.

Las máquinas de estado son redes de Petri en las cuales cada transición tiene exactamente una entrada y una salida. Resta por demostrar que es posible pasar de una red de Petri general a un autómata de estado. Para ello basta con demostrar que las transiciones que producen conflictos (AND FUENTE Y AND SINCRONIA) pueden modelarse a través de un autómata. Al igual que en el caso del grafo ICN, el AND SINCRONIA puede ser modelado a través de un autómata de estados en el cual si se tienen k arcos que deben sincronizarse, se requieren $2k$ estados. Cada estado refleja las posibles combinaciones de condiciones requeridas para continuar tras el punto de sincronismo. El AND FUENTE puede obtenerse con transiciones vacías.

2 ADMINISTRACIÓN DE LA INFORMACIÓN

Como hemos visto antes, un flujo se compone de una serie de actividades, las cuales pueden o no suponer interacción con participantes, donde estos pueden ser individuos, grupos, o recursos (impresoras, alarmas, maquinaria semi-automática, etc.), dependiendo del flujo en cuestión. Lo que si es claro es que en general, todo flujo, consta de una determinada data que viaja a través del mismo y es modificada en algunas de las actividades que lo componen.

En tal sentido, nos hemos visto en la necesidad de definir como se administrará dicha información. El caso más claro, está dado en una actividad que requiera la interacción con el usuario. En este tipo de actividades, al usuario se le debe presentar el estado actual de la data, permitirle modificarla y luego incorporar los cambios que éste haya hecho. Todos estos temas son independientes del mecanismo de almacenamiento, estamos aquí enfocados no a la administración del almacenamiento físico, ni a seleccionar un medio, sino a un concepto más general, estamos encaminados a plantear un tratamiento genérico y efectivo de la información.

Una primera opción podría haber sido dejarla libre, definir un procedimiento de almacenamiento adecuado y luego diseñar una página web, para cada instancia (actividad del flujo) que requiriera una interacción con usuarios. La primera y más clara de las desventajas, está dada por la constante necesidad de creación de páginas nuevas, estructuras de almacenamiento (estructuras de base de datos, estructuras de directorios para almacenar archivos, esquemas de XML, etc., de acuerdo a los mecanismos seleccionados en cada ocasión). Si la decisión fuese esta, cada proceso de instalación de un Workflow en una determinada organización, supondría hacer un relevamiento de todas las ParticipantTask de cada uno de los flujos, y para cada una de ellas, diseñar una página web adecuada. Otra dificultad implícita en esta propuesta es que no se tiene una certeza del formato de la información a administrar, con la subsiguiente pérdida de generalidad en el tratamiento de la misma. Más aún, si pensamos en interactuar con procesos, no con usuarios, es muy importante pensar una forma de administrar la información genérica, dúctil y utilizable.

Dado que la anterior no era una buena propuesta, decidimos buscar una forma de almacenar la data que fuera suficientemente estructurada como para permitir un manejo genérico de la misma y suficientemente amplia, dúctil y flexible como para permitir almacenar cualquier tipo de data que podamos encontrar en un flujo de trabajo. Claramente, la forma más estructurada y auto descriptiva, a la vez que libre, es almacenar la información en documentos XML.

Una vez definido que el almacenamiento e intercambio de información entre las distintas tareas del flujo se haga mediante documentos XML, se comenzó a pensar en las bondades que tendría, tener un esquema ya definido, que fuera suficientemente potente para satisfacer las necesidades de los flujos y suficientemente definido para facilitar el tratamiento genérico por parte del Workflow.

Este pensamiento fue evolucionando hacia la idea de definir una abstracción, que vaya un paso más allá del almacenamiento físico de información y encare la misma de forma genérica, lo que terminó en la necesidad de definir esquemas a los cuales respondan los documentos XML.

2.1 XML – Esquemas XSD – Esquemas Relacionales -

Es claro que la data, se va a almacenar en alguna forma. Parece obvio también que la forma más común de almacenar data sea mediante bases de datos relacionales, a la vez que la forma más utilizada para compartir o intercambiar data entre distintos procesos de una misma organización (o incluso entre organizaciones) son los documentos XML. Estos motivos son más que suficientes para pensar que nuestra "estructura de datos" debe ser fácilmente mapeable a estos dos mecanismos de almacenamiento.

Todo documento XML, tiene un esquema XSD al cual responde. A la vez, generalmente (asumiremos que siempre) existe una estructura relacional (esquema relacional) en la cual se puede almacenar de forma adecuada los datos en una base de datos. Sobre tales supuestos, no parece ser imponer demasiadas restricciones a los datos a administrar, el hecho de que sean almacenables bajo una estructura relacional. Menos aún, si logramos que el almacenamiento de la información sea fácilmente almacenable tanto en XML,

como en bases de datos relacionales. La idea es generar un mecanismo de almacenamiento propio, que sea muy simple de manejar por el WorkFlow, a partir del cual podamos obtener el documento XML, el esquema al que éste responde, la representación relacional de los datos y sobre el cual el WorkFlow pueda hacer consultas, actualizaciones y demás, delegando la mayor parte del trabajo posible a herramientas externas al mismo.

DataSet:

Los conjuntos de datos (DataSets) almacenan datos en una memoria caché desconectada. La estructura de un conjunto de datos es similar a la de una base de datos relacional; exponiendo un modelo jerárquico de tablas, filas y columnas. Además, contiene restricciones y relaciones definidas para el conjunto de datos. Por tanto, es tan potente como una base de datos relacional y nos brinda la facilidad de trabajar desconectados de la misma y hacer actualizaciones periódicas, con un buen control de concurrencia previsto, mediante un modelo de objetos prediseñado en el Framework de .NET.

Un conjunto de datos de ADO.NET es una vista relacional de los datos que se pueden representar en XML. En Visual Studio y .NET Framework, XML es el formato estándar para almacenar y transmitir datos de todo tipo. En sí, los conjuntos de datos tienen una estrecha relación con XML. La estructura de un conjunto de datos, sus tablas, columnas, relaciones y restricciones, puede definirse en un esquema XML (un XSD). Los esquemas XML son un formato basado en estándares de W3C (World Wide Web Consortium) para la definición de la estructura de datos XML.

Por su naturaleza los conjuntos de datos, deben ser llenados con datos provenientes de un origen (puede ser una base de datos o documentos XML), permiten la administración libre de los datos (inserción, modificación y eliminación), para luego impactar estos cambios en una base de datos o en una nueva versión del documento XML de origen. El DataSet, en combinación con la otra gran vedette de ADO.NET, el adaptador de datos (DataAdapter), resuelve gran parte la complejidad de leer, presentar, modificar y actualizar datos desde y hacia un dispositivo de almacenamiento.

Las anteriores propiedades, así como otras tantas que pueden analizarse en la sección de estudio de .NET, en su sección de ADO.NET, hacen del DataSet un aliado ideal a la hora de definir y administrar la información que viajará a través de cada uno de nuestros flujos. En tal sentido, **se ha definido que nuestros flujos administrarán la información en formato de conjuntos de datos (DataSet)**. Yendo un poco más allá, intentamos encapsular en el concepto de "Esquema" algo más que el DataSet, pero siempre manteniéndolo como la estructura básica de nuestro mecanismo de administración de información. Los DataSet brindan una colección genérica de propiedades, almacenadas internamente como un HashTable, donde se pueden definir propiedades del tipo {Clave,Valor} y luego, buscando por la clave obtener el valor asociado a esa propiedad para un determinado DataSet. La misma colección de propiedades está presente en las tablas que lo integran, como también en las columnas.

Administración de Esquemas.

Una vez decidido que se administre la información mediante el concepto de esquema, veamos que funcionalidades deseamos darle para que se puedan automatizar la mayor cantidad de etapas que supone la administración de la misma. Nos proponemos automatizar dos etapas fundamentales del proceso que son, la presentación de formularios a los usuarios que permitan visualizar y modificar información y también, en caso que queramos trabajar con bases de datos, tener una herramienta general de actualización de los cambios experimentados en una determinada instancia de un esquema, es decir, un documento XML.

Para satisfacer nuestras expectativas, necesitamos básicamente, tres herramientas que hemos llamado:

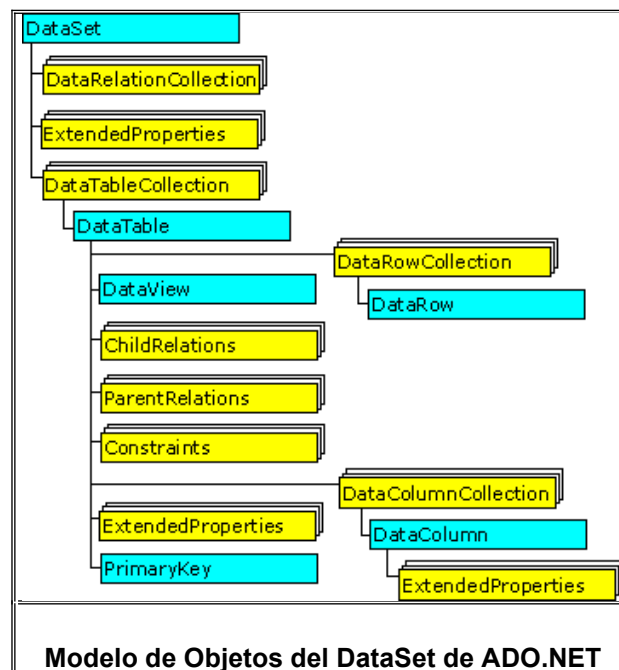
- **SchemaWeb:** Es deseable que el WorkFlow no tenga la responsabilidad de desplegar formularios y administrar información propia de los procesos particulares, ni dependiente de las aplicaciones. Esta tarea, la delega al SchemaWeb, que es la herramienta que, a partir de la estructura de un esquema, genera un formulario dinámico que permita al usuario visualizar, modificar, ingresar o eliminar información.

- **SchemaImpact:** Para los casos en que el almacenamiento se realice en bases de datos relacionales, que suponemos serán la gran mayoría, desarrollamos una herramienta que, basada en determinadas propiedades de los esquemas, impactarán los cambios en la fuente.
- **SchemaDefinition:** Permite definir un esquema, tanto en su estructura básica, como en una serie de propiedades circundantes que facilitan la tarea de las otras dos herramientas.

De lo anterior se deduce que el concepto de esquema engloba, no solo la definición de la estructura relacional, sino también las propiedades extra necesarias para resolver el trabajo de las dos herramientas genéricas de visualización e impacto en base de datos. En la siguiente sección, presentaremos la estructura de un esquema.

Estructura del esquema.

El esquema, como hemos manejado, supone la definición no solo de la estructura relacional, sino también de una serie de propiedades, tanto generales del esquema, como particulares de las tablas, columnas o relaciones que lo integran. La estructura de un DataSet, responde al siguiente modelo de objetos.



Básicamente todas las propiedades extras que necesitamos definir para sustentar el funcionamiento de las distintas herramientas, es almacenado como propiedades extendidas (ExtendedProperties) del objeto más adecuado. Dependiendo de lo que estemos intentando almacenar, podemos asociarlo como propiedades extendidas del DataSet, de las tablas, de las columnas o de las relaciones.

El esquema se compone de tablas y éstas de columnas y relaciones. Por tanto, la edición de un esquema supone la edición de Tablas, Columnas y Relaciones. A su vez, cada una de éstas tiene una serie de propiedades asociadas, útiles para la herramienta de definición, el editor de formularios, o el impactador automático de DataSets en base de datos.

- Tablas: Una tabla tiene las siguientes propiedades:
 - Propiedades de uso común a todas las herramientas:
 - Nombre: nombre de la tabla en la estructura del DataSet.
 - Descripción: Una descripción técnica de la misma, su significado en el esquema, etc., información destinada a registrar el objetivo de la creación de la tabla, o características particulares que se deseen perdurar.
 - Cardinalidad: Indicador de la cantidad de registros esperados para la tabla, en una instancia del esquema. Los valores válidos son:
 - 1: Indica que la tabla tendrá un único registro.
 - N: Indica que la tabla tendrá al menos uno, pero posiblemente muchos valores válidos.
 - *: Indica que la tabla tendrá un número no determinado de registros, incluso ninguno (a diferencia del caso anterior).
 - Clave primaria: Colección de campos de la tabla, que forman la clave de la misma. El concepto es análogo al de clave en una estructura relacional. Será utilizado por todas las herramientas del proyecto.
 - Propiedades visuales:
 - Visible: Indicador booleano de que la tabla será visible para el usuario. Es por tanto, análogo al concepto de modificable. Si el usuario la puede ver, la puede modificar. En general, este indicador estará en verdadero en la mayoría de las aplicaciones, pero se deja la posibilidad de guardar en el esquema información extra que no quiera ser presentada al usuario. Una extensión posible a la implementación actual del esquema es permitir que ciertos campos, tomen el estilo de "combo de selección", para lo cual puede ser útil tener los datos en una tabla oculta en el DataSet. Otra aplicación puede ser, guardar en el esquema información propia del proceso, dependiendo de la aplicación, pero no presentada al usuario. Por ejemplo, si el proceso llamará procesos externos al flujo, éstos pueden requerir el pasaje de información para su ejecución y esta ser pasada por este medio a las distintas aplicaciones.
 - Ayuda/ToolTip: Texto de ayuda al usuario final, indicando la utilidad de la tabla. Seguramente la edición de formularios, sepa que hacer con el valor de este campo. Una posibilidad amigable para el usuario es que cuando se posiciona sobre la tabla, le muestre este texto de ayuda.
 - Formularios: El objetivo de la posibilidad de definir formularios específicos para un esquema es mantener la extensibilidad y personalización de las aplicaciones. El usuario administrador del Workflow, podría ingresar el nombre de un control que quiere que se presente al usuario cuando llega una instancia determinada. Los formularios previstos para ingresar son:
 - Display: Formulario personalizado para desplegar el contenido, de forma inmodificable para el usuario.
 - Insert: Formulario personalizado para ingreso de nuevos registros.
 - Update: Formulario personalizado para la actualización de valores.
 - Delete: Formulario para seleccionar un registro a eliminar.
 - Propiedades relacionadas con la fuente de datos: En esta categoría se asocian principalmente las propiedades del esquema que serán utilizadas por el impactador en base de datos de los cambios en el contenido del mismo. Ellas son:
 - Persistente: Indicador de si el campo es o no persistente. En realidad, la intención es utilizarlo como indicador de si debe o no impactarse en la fuente de datos, los cambios que se realicen sobre estos campos.
 - Nombre Físico: En caso de ser persistente, se pretende actualizar la fuente con los cambios que se realicen, sea por parte de usuarios o procesos, en la fuente de datos. Para ese caso, nos interesa saber cual es el campo físico al que mapea la columna actual.
 - Proceso de Insert: Es el identificador del proceso que llevará a cabo la inserción de registros nuevos que se creen sobre esta tabla, en tiempo de ejecución.

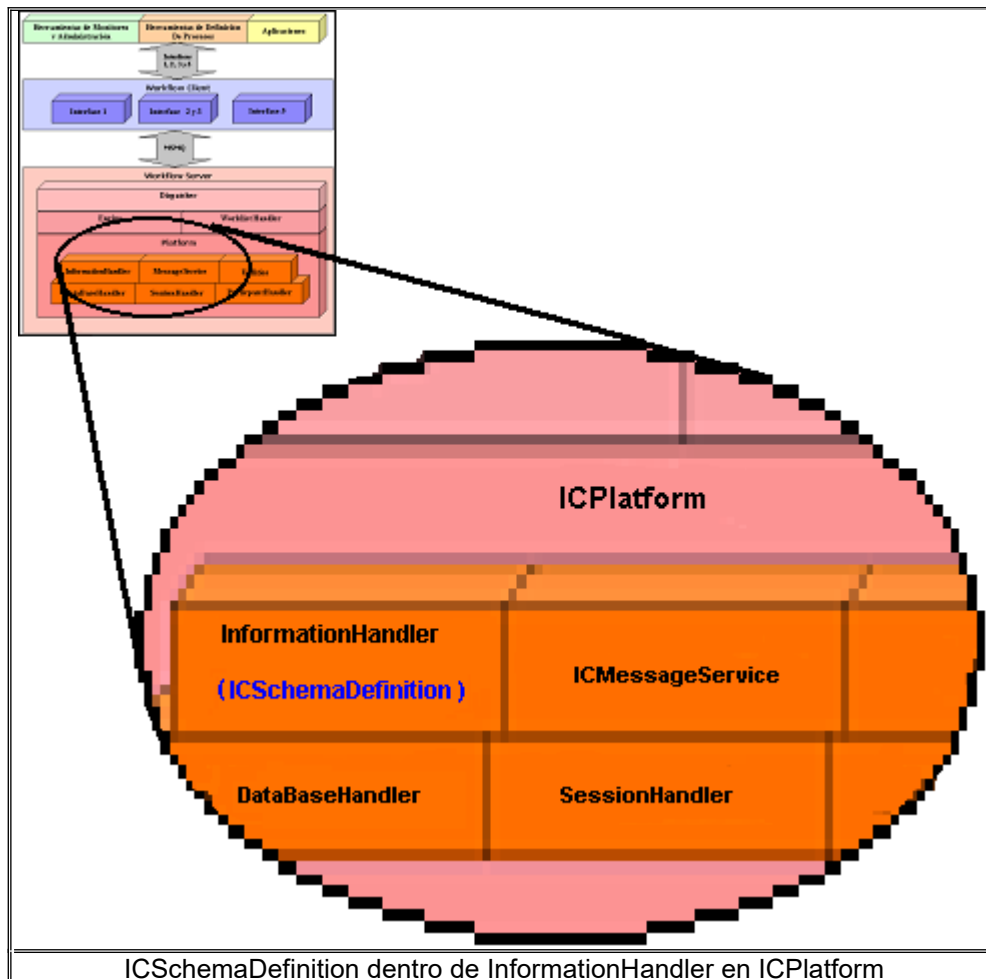
- Proceso de Update: Es el identificador del proceso que llevará acabo la actualización de registros modificados en tiempo de ejecución.
- Proceso de Delete: Es el identificador del proceso que llevará acabo la eliminación de registros que se eliminen de la tabla, en tiempo de ejecución.
- Columnas: Una columna consta de las siguientes propiedades:
 - Propiedades generales:
 - Nombre: Nombre con que se identifica la columna en el esquema.
 - Tipo: Tipo de los datos que almacena.
 - Modificable: Indicador de si debe o no permitírsele al usuario modificar el dato que esté almacenado en este campo.
 - Tamaño: Es la cantidad de dígitos o caracteres reservados para almacenar la información (en caso de que el tipo lo requiera, por ejemplo, entero, decimal, string)
 - Lugares decimales: Para el caso del tipo de datos decimal, indica la máxima cantidad de dígitos posteriores a la coma que se permitirá almacenar.
 - Descripción: Una descripción técnica, destinada al usuario administrador, no al usuario final, que se desee conservar, relacionada con el campo que se define.
 - Validaciones: Hay tres tipo de validaciones previstas, para los datos acumulados en una instancia de un esquema.
 - Valor: La validación del valor ingresado por los usuarios, puede ser tanto por valor como por intervalo. Una validación por valor sería, por ejemplo, $x > 3$, mientras que por intervalo sería fecha $\geq 01/01/04$ y fecha $\leq 01/01/05$.
 - Tipo: Indicador de si la validación es por valor o por intervalo. Si el tipo de validación es por valor, se debe guardar un operador y un valor. En caso de seleccionar validación por intervalo, se deben seleccionar dos operadores e intervalos.
 - Operadores: Los operadores válidos son $\{=, <, >, <>, \geq, \leq\}$.
 - Valores: Los valores, deben ser ingresados por el usuario y guardados en el esquema para validar luego los datos ingresados, contra esta información.
 - Script en el cliente: Se ingresa aquí un string, con el código de un script que se ejecutará en el cliente web para validar los datos ingresados.
 - Script en el servidor: Se ingresa aquí un string, con el código de un script que se ejecutará en el servidor web para validar los datos ingresados.
 - Propiedades visuales: Los datos se presentarán visualmente en dos formatos. Cuando se presente la lista de valores de una tabla, se presentará como una grilla. Cuando se represente un único registro, se hará en forma de formulario para ingreso o presentación de datos.
 - Relacionadas a su presentación en una grilla:
 - Visibilidad: Indica si el campo es visible para el usuario en la grilla. En caso de ser este valor negativo, indica que cuando se despliegue la grilla mostrando los valores de una tabla, no se muestra esta columna.
 - Índice: Indica el índice de la grilla en el que se presenta el campo. No necesariamente debe coincidir con su posición en la tabla.
 - Nombre de Control: Esta propiedad permite establecer un nombre de control web que desplegará la información contenida en el campo, en una grilla.
 - Relacionadas a su presentación en un formulario:
 - Visibilidad: Indica si el campo será visible en formulario.
 - Índice: Indica el índice con el que se presenta en los formularios.
 - Nombre de Control: Esta propiedad permite establecer un nombre de control web que desplegará la información contenida en el campo, en una grilla.
 - Ayuda/ToolTip: Texto de ayuda que se presentará al usuario como guía de la utilidad del campo en cuestión.

- Propiedades relacionadas con la fuente de datos: Se agrupan en este ítem, una serie de propiedades que están directamente relacionadas con el objetivo de impactar automáticamente en base de datos, los cambios registrados en una instancia.
 - Persistencia: Indica si el campo debe o no ser persistido en la fuente de datos.
 - Admisión de nulos: Indica si se deben admitir valores nulos para estos campos.
 - Nombre físico: Indicador del nombre de columna en la fuente de datos, con la que mapea el campo en el esquema.

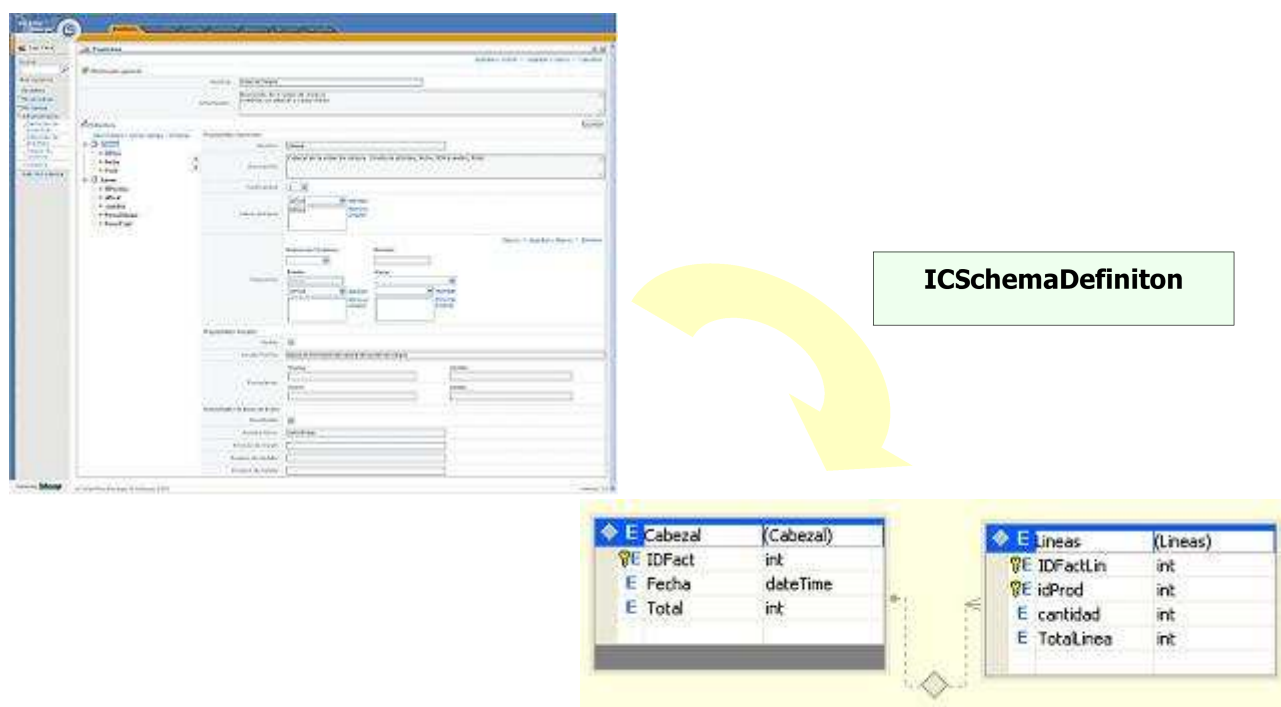
De aquí en más, definidas las principales propiedades del esquema, lo que queda en esta sección, es analizar cada una de las herramientas, como se valen de las propiedades para desarrollar su trabajo y la interacción entre éstas.

2.2 Definición de esquemas

El objetivo del ICSchemaDefinition es definir de forma genérica esquemas XSD que puedan ser utilizados con distintos objetivos tanto dentro del proyecto como fuera de él. Luego, tendremos cierta información que acumularemos en documentos XML, que respondan a esquemas definidos, por este medio, o en bases de datos, cuyo esquema relacional responda también al XSD correspondiente. Por ejemplo, a cada WorkItemInstance le asociamos un esquema al que responde la información que el usuario debe ver y modificar, en caso que sea necesario. Luego cuando el motor considere necesario mostrar un formulario con una cierta información a un usuario, buscará el esquema asociado al WorkItemInstance, para luego generar el formulario adecuado para visualizar la información de un XML que responda al esquema.



ICSchemaDefinition es, básicamente, una herramienta para definir esquemas, que es una parte de la definición de procesos de workflow. Por tal motivo, vemos al ICSchemaDefinition como una herramienta auxiliar a las herramientas de definición de procesos que manejan las interfaces de la WFMC. Es este el motivo de que la herramienta de definición de esquemas se comuniquen con el cliente de workflow (ICWorkflowClient) para operar, cuando estamos en el entorno de ICWorkflow. Ahora, por otro lado, se desea contar con la chance de ejecutar fuera del ambiente del Workflow, por lo que se han definido otros mecanismos de almacenamiento paralelos, independientes del Workflow, en archivos y base de datos.



Un esquema será definido por el usuario (probablemente administrador del sistema), a través de la herramienta web adecuada (ICSchemaDefinition). Ésta se comunicará con el cliente para invocar distintas funcionalidades que éste brinda, el cliente se comunicará con el servidor, a través del Dispatcher y éste con el motor del server (es quien debe brindar las funcionalidades de la interfaz uno). El motor guardará las definiciones de los procesos en base de datos, para tenerlas disponibles y poder ser consultadas, actualizadas y utilizadas.

ICSchemaDefinition lo hemos definido como una dll con las funcionalidades necesarias y una interfaz gráfica web adecuada para utilizar todo su potencial. Esta herramienta utiliza el cliente únicamente para las operaciones relacionadas con el almacenamiento, recuperación y modificación de esquemas, no para resolver problemas propios de la lógica de la administración de esquemas; lo que forma parte de la propia herramienta. Un objetivo subyacente a este módulo del proyecto es que la herramienta de definición de esquemas tenga un diseño tal, que sea fácilmente independizable del Workflow, en caso que se desee utilizar para un fin independiente de éste. Por tal motivo lo hemos diseñado como un proyecto independiente dentro del Platform (plataforma de herramientas genéricas utilizables en otros proyectos, no solo en el Workflow) por un lado, y la interfaz web que lo utiliza dentro del proyecto web que aglomera todas las interfases del Workflow. La dll como tal, deberá soportar el trabajo, integrado o no al Workflow.

2.3 Requerimientos:

El objetivo básico del ICSchemaDefinition es permitir la definición de esquemas XSD. Por lo expuesto anteriormente, se ha resuelto que la definición de esquemas, se haga en realidad, basándonos en un DataSet. Por tanto, el objetivo real del ICSchemaDefinition pasa a ser permitir al usuario definir la estructura del DataSet (conjunto de datos) que desea definir. Debe brindar la posibilidad de definir completamente la estructura del DataSet, a la vez que debe brindar una serie de potencialidades adicionales, almacenadas como propiedades extendidas, cuya utilidad ha sido explicada previamente, en la sección en que se presenta el concepto de esquemas. Es requisito, según lo definido por los usuarios, que permita visualizar en forma adecuada (posiblemente como un árbol) la estructura parcial de los esquemas mientras se definen, a la vez que tanto a nivel general, como de tablas y columnas, permita definir una serie de propiedades que son básicas para poder explotar al máximo las funcionalidades del conjunto de datos de .NET Framework.

Las funcionalidades que debe brindar la herramienta son las siguientes:

- Comunicación con el cliente para guardar y recuperar esquemas. En este punto contemplaremos también generar repositorios de esquemas en bases de datos independientes del Workflow y en archivos de definición de esquemas tradicionales (XSD), como forma de dejar libre la independización del Workflow.
- Administración de propiedades.
 - Cambiar el orden de visualización de tablas y campos en el árbol.
 - Administrar nombre y descripción del Esquema.
- Administración de la estructura del Esquema.
 - Administración de Tablas.
 - Alta, baja y modificación de tablas.
 - Administración de propiedades de las mismas.
 - Administración de Campos, dentro de las tablas.
 - Alta, baja y modificación de campos o columnas.
 - Administración sus propiedades.
 - Administración de Claves Primarias de las tablas.
 - Alta, baja y modificación de claves primarias.
 - Administración de Relaciones entre Tablas (Foreign Keys).
 - Alta, baja y modificación de relaciones.
 - Consultas sobre las relaciones (tabla hija en la relación, columnas que conforman la relación, etc.).

2.4 Diseño.

La solución consta de una interfaz gráfica adecuada y una capa lógica que, básicamente en función de un DataSet que utiliza como base, expone una serie de operaciones útiles para la definición del esquema. El diseño general de esta parte de la solución es respetando el estilo empleado en el resto de las soluciones desarrolladas en el ambiente de la empresa, en el que las interfaces web se agrupan todas en un único proyecto web por solución. Por tal motivo, la página web está integrada al resto de la solución web del proyecto.

Interfaz Gráfica (Web):

Se necesita desarrollar una única interfaz web, que permita al usuario definir el esquema de forma adecuada, a la vez que ir observando como va quedando el esquema. Una primera opción hubiera sido definir una página web a tales efectos, que en .NET se hace desarrollando un formulario web, que se guarda como aspx, que identifica a las páginas ASP de .NET. Se ha ido un paso más allá, y hemos resuelto nuestro problema mediante un UserControl de .NET. En pocas palabras, un UserControl tiene las mismas funcionalidades que un formulario web (WebForm), pero luego se puede insertar de forma muy sencilla en cualquier página web (más aun, una página web puede contar de un único control). Para desarrollar UserControl, en el espacio de nombres "System.Web.UI" (interfaces web del Framework), cuenta con una clase llamada UserControl. Debemos crear una clase propia derivando de ésta e implementando los métodos establecidos. Básicamente hay que definir el método page_Load (cuando se carga la página se llama a este evento de cada control en la misma) y luego los eventos asociados a los controles (botones, combos, listas, árboles, etc.) que se agreguen al control desarrollado.

El diseño de la página responde básicamente a los requerimientos, teniendo una zona superior para definir las propiedades generales del esquema y luego dos zonas inferiores, una donde vemos un árbol que representa la estructura parcial del esquema y otra donde se editan las propiedades de tablas y columnas, seleccionadas en el árbol anteriormente citado.

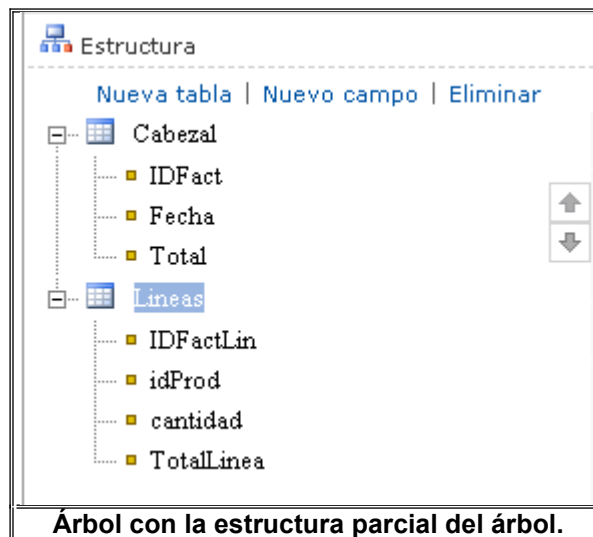
Visión general del SchemaDefintionForm.

La anterior es una presentación del formulario web cargado, con la edición de un esquema de una orden de compra. Vemos claramente los sectores de información definidos. A continuación se presenta y comenta cada uno de ellos.

Edición de cabezal, propiedades generales del esquema.

La imagen anterior, presenta con un detalle un poco mayor el sector superior del formulario, destinado a la definición de las propiedades generales del esquema. Permite definir nombre y descripción.

En la parte inferior, a la izquierda se presenta un árbol, donde el usuario observará el estado actual del esquema, la estructura del mismo, sus tablas y columnas. En el ejemplo, observamos el estado de una ejecución parcial de la creación de una orden de compra.



En la sección inferior derecha se editan las propiedades de las tablas o campos, según se esté ante la edición de una u otra. Lo que define es la selección realizada sobre el árbol de la izquierda. A continuación se presenta una imagen de la edición de la tabla de las líneas.

Propiedades Generales:	
Nombre:	Lineas
Descripción:	Lineas de la orden de compra.
Cardinalidad:	1
Clave primaria:	IDFactLin Agregar
	IDFactLin Eliminar idProd Limpiar
Relaciones:	Relaciones Foráneas: Nueva Guardar y Nueva Eliminar Desde: CabezalLinea Agregar Eliminar Limpiar Hacia: IDFact Agregar Eliminar Limpiar IDFactLin Agregar Eliminar Limpiar
	Nombre: CabezalLinea
	Desde: IDFactLin
	Hacia: IDFact
Propiedades Visuales:	
Visible:	☒
Ayuda/Tooltip:	Ingrese la información de las líneas de la orden de compra.
Formularios:	Display:
	Update:
	Insert:
	Delete:
Propiedades de Base de Datos:	
Persistente:	☒
Nombre físico:	LinOrdComp
Proceso de Insert:	
Proceso de Update:	
Proceso de Delete:	

Edición de las propiedades de la tabla "Líneas" de la orden de compra.

Edición de las columnas. A continuación se presenta la edición de la columna

Propiedades Generales

Nombre: IDFact

Tipo: Integer ☒ ¿Solo lectura?

Largo: Largo de Definición: 10

Descripción: Identificador de factura. Será un autonumérico, generado por el proceso de insert.

Propiedades Visuales

Grilla: ☒ ¿Visible? Índice: 1 Nombre de Control:

Formulario: ☒ ¿Visible? Índice: 1 Nombre de Control:

Ayuda/ToolTip: Identificador único de factura.

Propiedades de Base de Datos

Nulos: ☐ Persistente: ☒ Nombre Físico: CaboFact

Validaciones

Tipo: ☐ Ninguna ☒ Valor ☐ Intervalo Valor: >= 0

Script Cliente:

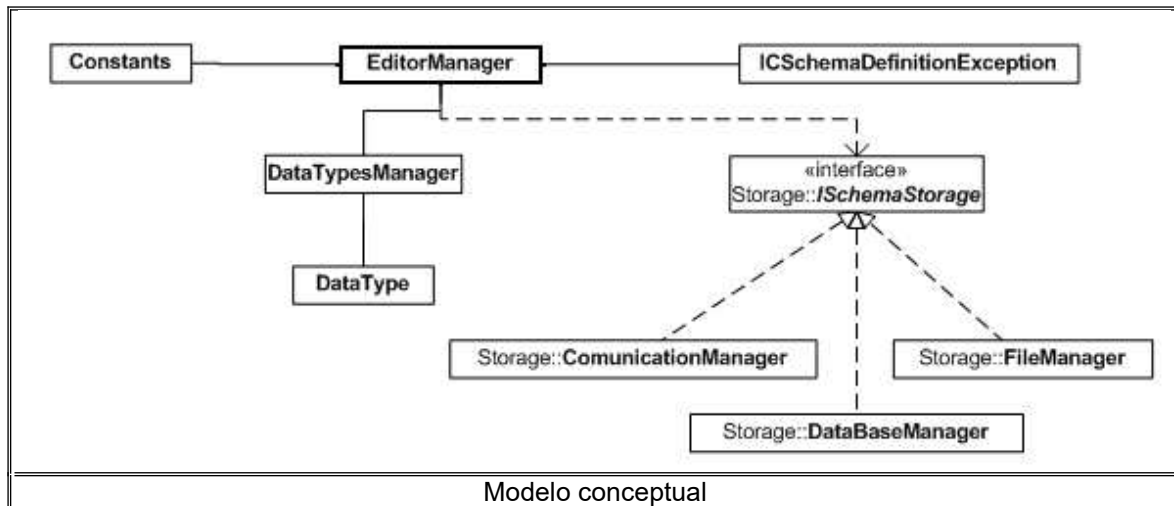
Script Servidor:

Edición de las propiedades de la columna "IDFact" del cabezal de la orden de compra.

2.5 Lógica de la Solución.

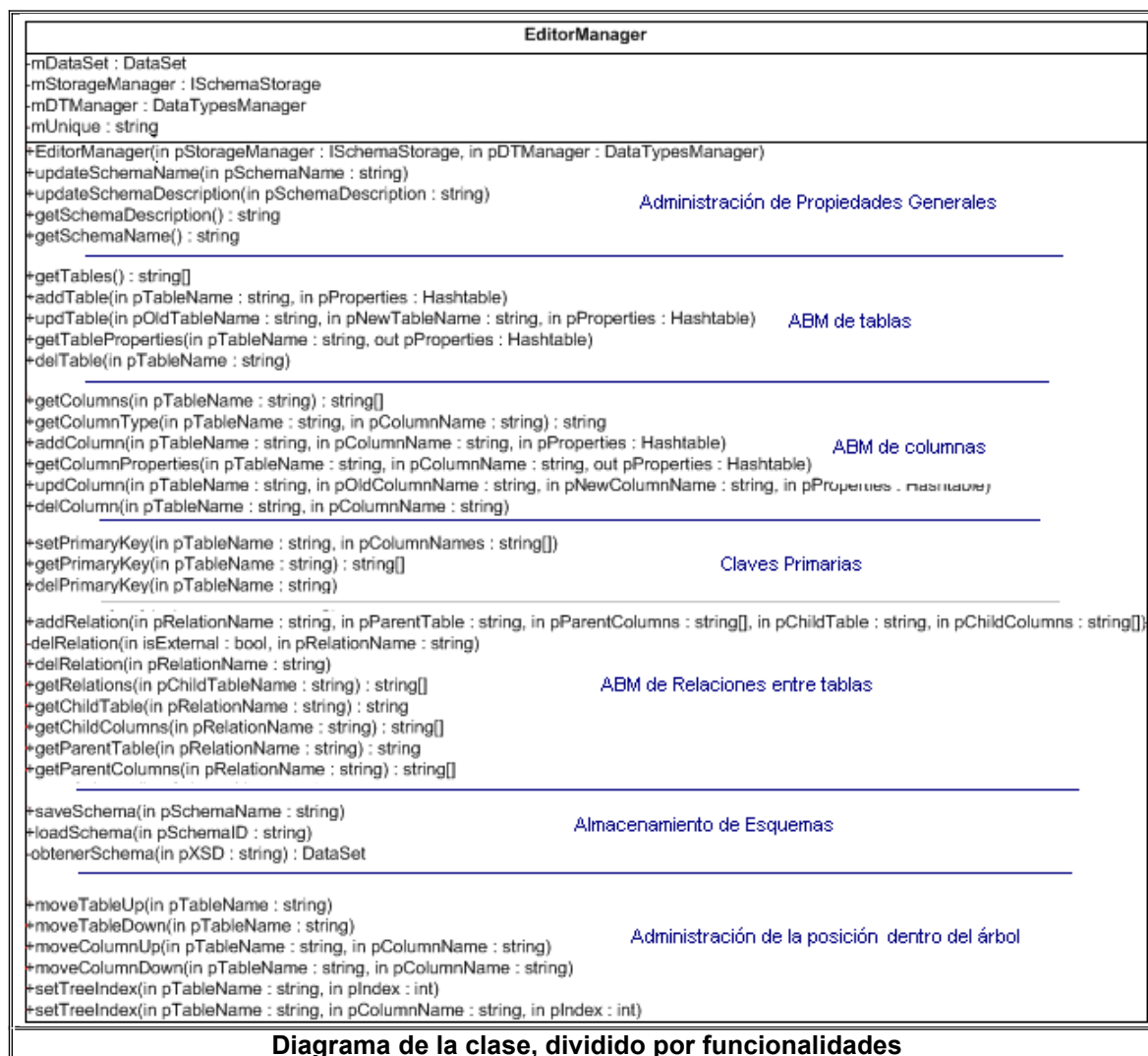
La solución básicamente se lleva a cabo en la clase EditorManager, que contiene la mayor parte de la lógica. El EditorManager es la cara visible de la dll, conteniendo como miembro fundamental un DataSet que utiliza como auxiliar para almacenar las definiciones que sobre el esquema, realice la interfaz, y deriva ciertas tareas, como la administración de tipos y el almacenamiento de esquemas a clases específicas para tales fines. Para el almacenamiento, se ha definido una interfaz que establece tres métodos (guardar un esquema, actualizarlo y presentar un esquema, dada una clave). Por el fin que llevó a la construcción del Editor de esquemas, hemos desarrollado una de las tres clases que implementan la interfaz, la que se comunica con el Workflow para delegarle las tareas de almacenamiento, listado y recuperación de esquemas. Se han definido dos más a modo de ejemplo, una que los almacenaría serializados en una base de datos externa y otra que directamente los almacena mediante archivos de esquema (extensión XSD).

Otro problema a resolver es el mapeo entre tipos de datos "lógicos", es decir, los tipos de datos utilizados por los usuarios (texto, fecha, hora-minuto, decimales, enteros, etc.) y los tipos de datos que permite almacenar un campo de una tabla de un DataSet y un campo físico en un posible base de datos externa.



Administración de estructura y propiedades (clase EditorManager).

La clase EditorManager es la que resuelve la definición de esquemas brindando todas las funcionalidades adecuadas, tal como se ha analizado en la sección de requerimientos. Para hacerlo, se vale de la colaboración de las clases CommunicationManager (en el caso particular del Workflow) y DataTypesManager, como vimos antes. La clase, además de una referencia a un objeto de cada una de las clases anteriormente citadas, tiene un miembro del tipo DataSet, sobre el cual efectúa los cambios (agregar, modificar, eliminar, etc.) que le indique la interfaz y un string llamado mUnique, que contiene el identificador único asignado a la definición del esquema que estamos manejando.



Administración de propiedades: Las propiedades generales del esquema son nombre y descripción. El EditorManager expone métodos para consultar y actualizar ambas propiedades del esquema.

Administración de la estructura del Esquema:

Esta sección agrupa todos los métodos que se utilizan para la administración de la estructura del esquema, es decir administración de tablas, columnas, claves y relaciones.

Administración de Tablas: Hacer alta, baja y modificación de tablas. Administra la estructura y las propiedades de las mismas. A cada tabla asociamos una serie de propiedades extendidas (ExtendedProperties que almacena el DataSet) con una serie de valores de interés para nosotros. Esos valores son, por ejemplo, el nombre del proceso que realiza la actualización en la fuente de datos, el que borra, el que agrega, etc. Otras propiedades pueden ser el tooltip que en el formulario le aparecerá al usuario cuando se posicione sobre esta columna, el nombre de la tabla en la fuente de datos con la que se corresponde, si es o no visible y que cardinalidad de filas manejará normalmente. La mayor parte de estas propiedades son para almacenar datos que serán útiles a futuros procesos en el Workflow, como por ejemplo, la generación automática de formularios a partir de un esquema.

Administración de Campos, dentro de las tablas: Hacer alta, baja y modificación de campos. Al igual que para las tablas, a las columnas le hemos asociado una serie de propiedades extendidas que permiten almacenar información interesante.

Administración de Claves Primarias de las tablas: En las estructuras relacionales de almacenamiento, hablamos de claves primarias asociadas a las tablas; de forma similar en los esquemas XSD, hablamos de restricciones de unicidad y, según la estructura que seleccionemos para nuestro XML, podemos generar esquemas con claves primarias. En nuestra visión de esquemas, las claves primarias se manejan con un sentido más similar al de las estructuras relacionales. Lo que hace el EditorManager, en pocas palabras es promulgar los métodos que expone el propio DataSet para la administración de claves, asociadas a las tablas. A nivel de interfaz, tendremos las tablas y columnas identificadas por nombres, mientras que los métodos expuestos por el DataSet, manejan objetos de su modelo de objetos (DataTable, DataColumn, DataRestriction, DataRelation, etc.), resolviendo el EditorManager, este mapeo.

Administración de Relaciones entre Tablas (Foreign Keys): De forma muy similar a lo que sucede con las Claves primarias, el EditorManager expone métodos que permiten hacer el alta, baja y modificación de relaciones entre columnas de distintas tablas. Que pendiente, por ser recortado del alcance, la definición de relaciones externas, es decir, relaciones que apunten a campos de tablas en otros esquemas, no el actual.

Cambiar el orden de visualización de tablas y campos en el árbol: Un requerimiento no funcional para esta aplicación, era que permitiera visualizar de forma simple y amigable una estructura en forma de árbol con las tablas y campos del esquema. La visualización en el árbol, como se ve en la sección referida a la interfaz gráfica, soporta cambiar el orden de las tablas y los campos, dentro del esquema. El soporte que, en la capa lógica damos a esta funcionalidad es que, tanto en las tablas como en las columnas, se agrega una propiedad extendida, utilizada para guardar el índice de visualización en el árbol. Sobre estas propiedades brindamos métodos de consulta y actualización.

Administración de tipos de datos.

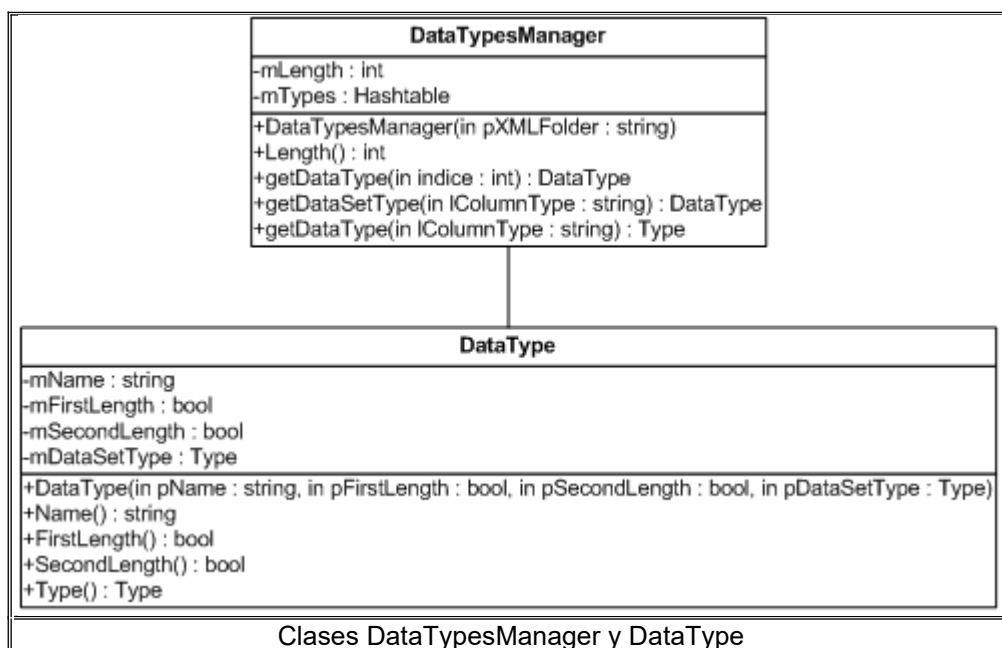
La clase DataTypesManager administra los tipos de datos que se permiten definir sobre nuestros esquemas. En un principio, se definen ocho tipos de datos que detallamos a continuación:

- Integer: Tipo de datos entero.
- Decimal: Tipo de datos decimal.
- Text: Texto largo.
- String: Texto medio.
- Date: Tipo fecha.
- DateTime: tipo fecha y hora.
- Time: Tipo hora.
- Boolean: Tipo booleano.

Esta colección de tipos de datos, es implementada mediante un hash. Este hash se carga con parejas clave-valor. Las claves, son los ocho tipos manejados anteriormente. A cada uno de ellos, se le asigna como valor un objeto de la clase DataType (clase auxiliar de la DataTypesManager). Un objeto DataType es un registro con cuatro valores:

- Nombre de tipo
- Indicador booleano sobre si se debe solicitar al usuario la definición de largo para campos de este tipo.
- Indicador booleano sobre si se debe solicitar una segunda definición de largo (por ejemplo para los decimales).
- El tipo de datos al que se mapea en la estructura del DataSet. Este será uno de los valores de la enumeración System.Type, de .NET Framework.

La clase `DataTypesManager` y su clase auxiliar `DataType`, son simples y se presentan a continuación.



Los tipos de datos a administrar, los obtiene de un documento XML llamado `DataTypes.xml` que se debe encontrar en la ruta que se pase por parámetro al constructor de la clase. La interfaz gráfica debe saber obtener de un lugar adecuado, por ejemplo del archivo de configuración de la aplicación, la carpeta adecuada donde encontrar el archivo. El XML original, con la definición de los tipos, es el que se observa a continuación, pero se pueden agregar tipos cuando se considere necesario sin afectar el funcionamiento de la aplicación.

```

<?xml version="1.0" encoding="utf-8" ?>
<types>
  <type name="Integer" FirstLength="YES" SecondLength="NO" DataSetType="Int32"></type>
  <type name="Decimal" FirstLength="YES" SecondLength="YES" DataSetType="Double"></type>
  <type name="Text" FirstLength="YES" SecondLength="NO" DataSetType="String"></type>
  <type name="String" FirstLength="YES" SecondLength="NO" DataSetType="String"></type>
  <type name="Date" FirstLength="NO" SecondLength="NO" DataSetType="DateTime"></type>
  <type name="DateTime" FirstLength="NO" SecondLength="NO" DataSetType="DateTime"></type>
  <type name="Time" FirstLength="NO" SecondLength="NO" DataSetType="DateTime"></type>
  <type name="Boolean" FirstLength="NO" SecondLength="NO" DataSetType="Boolean"></type>
</types>
  
```

XML con la configuración de tipos de datos.

Administración del almacenamiento (Interfaz `ISchemaStorage` y sus implementaciones).

En esta interfaz hemos definido tres métodos que debe implementar toda clase que quiera registrarse como administradora del almacenamiento de esquemas. Los métodos se detallan a continuación:

- **`insertSchema(DataSet pDataSet) : string`**: Debe guardar un `DataSet` en el medio específico de almacenamiento y retornar la clave que le ha sido asignada al mismo.
- **`updateDataStructure(string pIDDataSet, DataSet pDataSet)`**: Actualiza la estructura del conjunto de datos identificado por el primer parámetro en la fuente de datos, con la del conjunto `pDataSet`.
- **`getSchemaStructure(string pIDDataSet) : string`**: Dado un identificador de esquema, lo busca en el repositorio de datos y devuelve su serialización.

La interfaz puede ser implementada tantas veces como se desee. Nosotros hemos desarrollado tres implementaciones de la misma:

- **Clase DataBaseManager:**

Esta clase implementa la interfaz definida anteriormente, utilizando como repositorio de esquemas una base de datos. En este caso, el string de conexión a la misma, lo recibe por parámetro y en dicha base debe haber una tabla schemaDefinition, con un campo IDSchema y un SchemaXSD. Eso es lo que esta clase espera encontrar en la base de datos a la que se conecta.

- **Clase FileManager:**

Esta clase implementa la misma interfaz, utilizando como repositorio de esquemas un directorio definido en la parametrización de la aplicación. El constructor, recibe por parámetro una dirección donde está el repositorio de esquemas. En esa carpeta el FileManager, guardará los archivos de esquemas XSD, para recuperarlos luego.

- **Clase CommunicationManager:**

Esta clase implementa también la interfaz ISchemaStorage, encapsulando la comunicación con el cliente de WorkFlow. Es la que utiliza el sistema de WorkFlow, para almacenar los esquemas definidos en la base de datos interna. El cliente de WorkFlow expone métodos relacionados con el ABM de definiciones de esquemas. Por tanto, cada método crea un objeto de la clase ICWorkflowClient, para luego invocar al método adecuado del cliente. A continuación veamos un esquema de los métodos que expone la clase CommunicationManager para que utilice el EditorManager.

2.6 Editor de Formularios

El editor de formularios, a nivel de implantación de software está contenido en el proyecto SchemaWeb. El proyecto SchemaWeb supone, dentro del ICWorkflow, la generación dinámica de formularios adecuados para presentar o solicitar al usuario información contenida en un documento XML, a partir del conocimiento de su esquema. Por tanto, en definitiva, se puede pensar como la búsqueda de un formulario que "sepa" auto generarse a partir de la definición de nuestra extensión de un esquema XSD. Con esta herramienta, se busca simplificar todo el desarrollo de interfases de usuario. Cada vez que el Workflow deba interactuar con un usuario, será para exponer o solicitar información. Sea cual sea el caso, en las bases del proyecto se ha establecido que a cada instancia de interacción con el usuario, en tiempo de definición del flujo, se le debe asociar un esquema, al que responde la información almacenada. Por tanto, se puede asumir que en tiempo de ejecución, al llegar a una de estas actividades, se conoce de antemano el formato al que la información responde, lo que no es un tema menor. El objetivo del SchemaWeb es entonces, brindar una herramienta que, a partir del esquema, genere los formularios web adecuados para su visualización. La herramienta debe ser capas de, tomando un esquema, desplegar el formulario adecuado y, en caso de tener información asociada, presentarla en el mismo.

Creación de instancia de proceso - Paso 2 de 2

Cancelar | < Anterior | Guardar y Nuevo

Información general

Sistema: ICWorkflow

Proceso: Aplicación Orden de Compra

Datos de ingreso al proceso

IDFact:

Fecha:

Total:

Verificar | Ingresar

idProd	cantidad	Estado	TotalLinea
1100 Recepción de rodapiés	Suspendido	Escalado	05/03/2003
1102 Recepción de rodapiés	Suspendido	Escalado	26/03/2003
1101 Recepción de rodapiés	En ejecución	Ingresado	26/03/2003
1175 Recepción de rodapiés	Suspendido	Escalado	25/03/2003
1174 Recepción de rodapiés	Suspendido	Escalado	25/03/2003
1153 Recepción de rodapiés	Suspendido	Escalado	25/03/2003
1142 Recepción de rodapiés	Suspendido	Escalado	25/03/2003
1143 Recepción de rodapiés	Suspendido	Escalado	25/03/2003

ICSchemaWeb_Editor

E Cabecal (Cabecal)

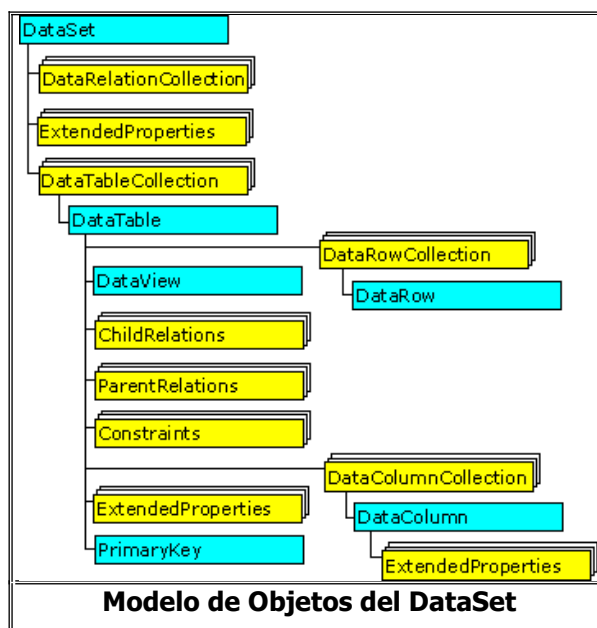
IDFact	int
Fecha	dateTime
Total	int

E Lineas (Lineas)

IDFactLin	int
idProd	int
cantidad	int
TotalLinea	int

Como se muestra antes, la idea es que el SchemaWeb utilice un esquema previamente definido mediante el SchemaDefinition, analizado anteriormente. El SchemaDefinition, es la herramienta que permite definir la estructura de un esquema en el cual se almacenará información. El SchemaDefinition, realiza este trabajo basado en nuestra definición de esquema (extensión del concepto de XSD, sustentado por el conjunto de datos o DataSet, provisto por ADO.NET), por lo que el SchemaWeb, generará el formulario, basado también en la estructura de un DataSet y sus propiedades extendidas para tablas y columnas, presentadas anteriormente como nuestro concepto de esquema. La idea básica de este proyecto web es que, a partir de una DataSet y sus propiedad extra, se pueda generar un formulario web (recordar que en la nomenclatura de .NET Framework, una página web se la denomina WebForm o formulario web), automáticamente, sin necesidad de retocarlo por parte de ningún usuario administrador, ni menos aun por un programador, para cada caso particular. Esto, conjuntamente con la definición genérica de esquemas, permite que la implantación de un nuevo flujo, suponga como disponible la definición de los esquemas a los que responde la información administrada en el mismo. Dicha definición podrá ser especificada con la herramienta SchemaDefinition o no, pero debe respetar la nomenclatura utilizada por ésta. Con solamente eso y el nuestro editor automático de formularios, se tiene cubierta toda la presentación de información al usuario.

Estructura básica del DataSet: Un DataSet se compone, básicamente, de Tablas, Campos (de un determinado tipo), restricciones y relaciones. A continuación vemos el modelo de objetos del DataSet incluido en ADO.NET en la plataforma .NET Framework.



La idea es que un usuario en una determinada instancia de un proceso, debe visualizar el contenido de un documento XML, que responde a una estructura relacional preestablecida. Por ejemplo, se puede plantear que se desea visualizar el contenido de una factura, que desea ingresar un comprador a su sistema. Una factura básica, puede tener un cabezal y varias líneas, donde el cabezal se compone de un número de factura (identificador único de la misma), una fecha, el RUC del proveedor, una descripción de la transacción y el total de la compra. Por su parte, una línea puede suponer un numerador de línea (identificador único de la misma), cantidad de artículos, código del artículo, precio unitario, precio total de la línea.

Subyacentemente se tiene una relación entre el cabezal y las líneas que se representa de forma análoga a las existentes en una estructura relacional. Por tanto, estableceremos que la clave de la línea está conformada por identificador de factura e identificador de línea, a la vez que definiremos que la clave primaria del cabezal es el "IDFactura", la clave primaria de la línea es "IDFactura,IDLinea" y existe una relación 1...N entre el identificador de factura a nivel de cabezal y de línea. Esta es la definición que se haría de la estructura de la base relacional, por ejemplo. Lo mismo se debe establecer al SchemasDefinition para obtener esta representación de factura.

Objetivo: El objetivo del SchemaWeb, es llegar a algo similar a la estructura de una factura, de forma genérica. Con la información propia de la estructura del esquema, más algunas propiedades extra, como por ejemplo la multiplicidad esperada de registros en una determinada tabla, la herramienta debe generar la página adecuada para la visualización de la información de un XML que responda al esquema, o dicho de otra forma, de los datos contenidos en un determinado DataSet, con ese esquema relacional. La multiplicidad de datos esperados en una tabla, es básicamente, la cantidad de registros que la misma contendrá. En tal sentido, por ejemplo, la multiplicidad del cabezal será siempre uno, y la de las líneas será "n".

En definitiva, el trabajo del SchemaWeb, se reduce a resolver de forma adecuada los siguientes puntos:

- Presentar adecuadamente la estructura de todo un DataSet, donde pueden haber varias tablas y relaciones entre éstas, a partir siempre de una tabla de entrada y sabiendo el modo de edición (puede ser solo lectura o Display, modo de edición o Update, o modo de creación o Insert). Debe presentar la información de la tabla principal y, a partir de ésta, utilizando las relaciones ir presentando la información de las tablas relacionadas. Para cada una, utilizando la multiplicidad (una de las propiedades extendidas de una tabla, remitirse a la definición del esquema), decide si presentar una grilla (varios registros) o un formulario (un único registro con todos los campos en un orden adecuado). Se ha definido un control de usuario llamado SchemaWeb-Main, para resolver esta tarea, que delega a otros dos controles (SchemaWeb_Grid, SchemaWeb-Form) la labor de desplegar grillas y formularios de presentación.
- Presentar la información contenida en una tabla, en formato de grilla. Esta tarea la resuelve el control de usuario SchemaWeb_Grid. Cuando el main, debe desplegar una tabla, llamará a éste control, seteándole algunas propiedades, pero delegando buena parte de la tarea.
- Seleccionar un registro de la misma, para editarlo en un formulario. En general el usuario visualizará parte de los datos (definido por el administrador al crear el esquema), no todos en la grilla. Cuando selecciona un registro, se le debe brindar la posibilidad de verlo completo, en forma de formulario para observar campos ocultos en la grilla, modificar valores, etc. En definitiva, se debe generar un enlace entre la grilla y el formulario.
- Editar el contenido de un registro, con un formulario adecuado, utilizando controles básicos para cada campo (teniendo controles específicos por tipo de dato a exponer). Llegando al mayor límite de detalle, el usuario estará observando un registro particular, de una tabla seleccionada. Para esto, se debe desarrollar un formulario adecuado. Cada campo tiene un tipo, y cada tipo tiene sus particularidades, por lo que el trabajo de presentar un campo en particular, también se delega a controles específicos.
- Reflejar en el DataSet, los cambios realizados por el usuario, en el formulario web. Cada vez que se despliega un formulario, se hace en un modo. Los modos admitidos son "Display" o "Update". Por tanto, si se edita en modo de actualización, se deben reflejar en el DataSet los cambios hechos por el usuario. Con este fin, se agregan al final del formulario, los botones de "Guardar" y "Cancelar". El botón de Guardar, solo se visualiza cuando se esté ejecutando el formulario en modo "Update". Cuando el usuario presione este botón, el formulario, debe actualizar en el DataSet, los cambios en el formulario.

Resolviendo estos puntos, tendremos que el motor, al enfrentarse a una ParticipantTask, podrá buscar en la base de datos cual es el esquema asociado a esa tarea y llamar al SchemaWeb, pasándole el esquema (o DataSet) a partir del cual debe desplegar el formulario junto con información básica sobre el mismo, (como ser cual es la tabla principal) y en caso que se desee desplegar información, se debe pasar en la misma instancia del esquema (cargar el DataSet previamente).

A continuación, se extiende el estudio de la herramienta, tratando los siguientes puntos:

- Nociones General.
- Modelo Conceptual.
- Detalle de las clases más interesantes (SchemaWeb_Main, SchemaWeb_Grid, SchemaWeb_Form, Controles básicos)

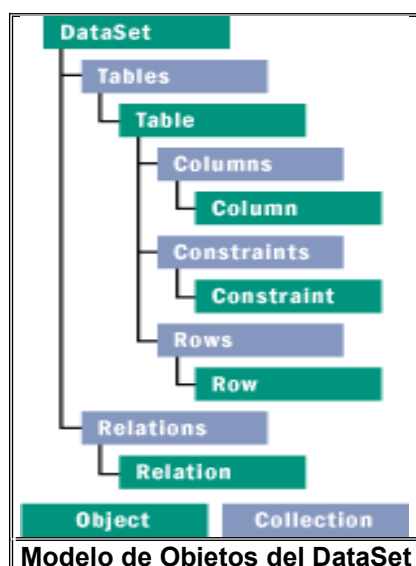
Nociones Generales.

Al invocar una página determinada en el navegador del cliente, se ejecuta el método Page_Load de la misma (evento levantado en el servidor previamente a la visualización), y luego, se disparan en cascada los Page_Load de los controles web de usuario que hayan sido agregados a la misma. Por tanto, en Page_Load de un control de usuario, se debe incluir todo aquel código que se desea ejecutar al momento de cargarlo y previamente a su visualización como parte de la página (o de otro control incluido en una de éstas). El mismo esquema se sigue al desplegar un control. Primero se ejecuta el Page_Load del mismo y luego en cascada, los Page_Load de los controles web que lo compongan.

Con este esquema, la idea es que para crear una página que presente un control de tipo SchemaWeb_Main, se agrega uno de ellos al contenido de la página. En el evento Page_Load de dicha página, se debe setear dos propiedades al control main. Las propiedades son el DataSet en base al que generarse y un indicador booleano de si mostrar o no el botón de impresión en el formulario resultante. Con solamente hacer esto, quedará automáticamente desplegado en pantalla, al llegar al cliente el formulario que responde al esquema pasado. Esto es posible porque, como manejaba antes, los eventos Page_Load, se llaman en cascada para los controles web de la página. Al agregar el control SchemaWeb_Main a la misma, posteriormente al load de la página se ejecutará el load del control. Por tanto, es en el load del control donde se deben agregar todos los "sub-controles", por decirlo de alguna forma, que conforman en control Main. A su vez, luego de ejecutado este load, se disparan los load de cada control dentro del main y así sucesivamente, se pueden construir controles tan complejos como se deseen. Ésta es la idea que hay detrás del diseño de esta solución y el Main lo que hará, será recorrer la información que deba presentar e ir agregando dentro suyo, controles de tipo grilla o formulario, según sean necesarios para representar el esquema. A su vez, por ejemplo, el control de tipo formulario (SchemaWeb_Form) se compondrá de un control por cada campo del registro que quiera editar.

Modelo Conceptual.

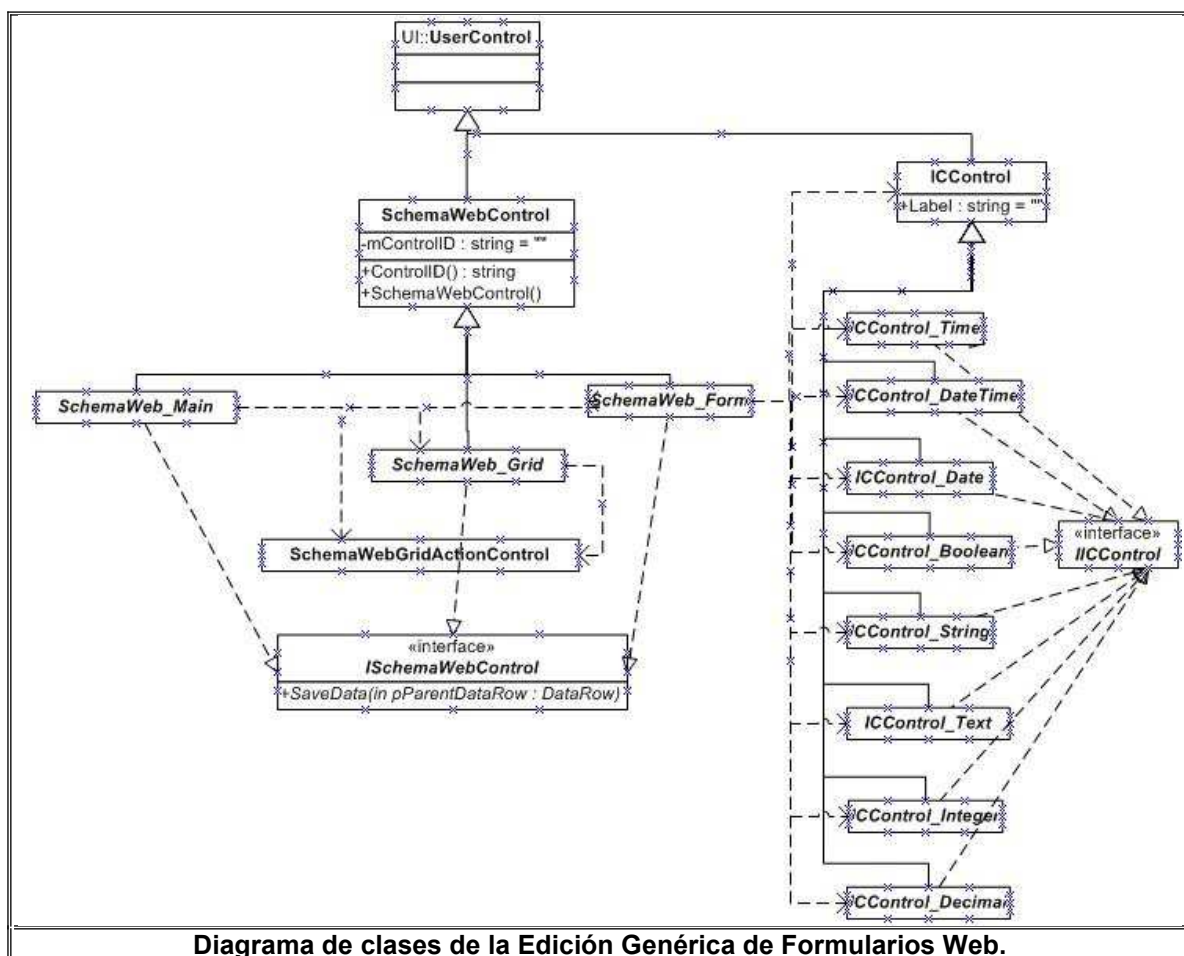
El modelo conceptual surge fácilmente de lo expuesto anteriormente. También se deriva sin mayor dificultad, a partir del modelo de objetos del DataSet (objeto de ADO.NET que da soporte a toda la solución presentada para la administración de la información). El modelo de objetos del DataSet, es simple y se presenta a continuación.



El modelo de objetos de nuestra solución sería, a cierto nivel de abstracción, muy similar al del DataSet, ya que el objetivo es que presente gráficamente de una forma adecuada uno de éstos. La solución presentada, tiene un objeto, el SchemaWeb_Main por medio del cual representa gráficamente un DataSet completo. A partir de una tabla, y navegando a través de sus relaciones, presentará cada una de las tablas al usuario. Para presentar una tabla, utiliza el objeto SchemaWeb_Grid (grilla). A su vez, una tabla consta de una serie de registros. Cuando se desea presentar más en detalle el contenido de un único registro, se lo hace mediante un tercer objeto, el SchemaWeb_Form. Un registro, es una colección de campos, con valores asociados. Cada uno de éstos campos, almacena información de un determinado tipo. La solución gráfica, por tanto, cuando presenta un registro por medio del control que edita un formulario, utiliza distintos tipos de controles individuales para presentar los distintos tipos de campo (en función del tipo de datos).

Si a lo expuesto, se le agrega el concepto de acciones que el usuario puede ejecutar sobre un grilla, como ser seleccionar un registro para editar (en modo visualizar o modificar), o eliminar un registro de la misma, se debe agregar cierta inteligencia a la misma. El control `SchemaWeb_Grid`, cuenta con una colección de objetos que representan estas acciones y se manifiestan gráficamente como columnas extra con botones cuya acción asociada supone la llamada al formulario adecuado, con los parámetros apropiados. La clase `SchemaWebGridActionControl`.

Agregando algún concepto más, como ser que todos los controles elementales heredan de una clase genérica `ICControl`, que agrupa funcionalidades comunes de validación de largos y tipo de la información ingresada, como también la etiqueta presentada a la izquierda del control en un formulario, a la vez que implementan la interfaz `ICControl`, que engloba una serie de propiedades comunes a todos los controles como ser el identificador único, el tooltip, valor, largo máximo de la data, porcentaje del ancho total del control, etc, se llega a una solución basada en el siguiente diagrama de clases.



Controles básicos.

Los controles básicos son los que se observan a la derecha en el diagrama de clases anterior y existe uno por cada tipo de datos a almacenar, en el estado actual de la solución. Era la intención que la administración de la información quedara abierta a seguir creciendo y una de las direcciones en las que se prevee crecer en la cantidad de tipos de datos a almacenar. En forma paralela, se podrían implementar distintos controles para presentar un mismo tipo de datos.

Para cumplir con estos dos objetivos es que el tipo de datos a administrar se obtiene de un documento XML, por lo que agregar un tipo de datos, se reduce a agregar una línea al documento. Si se quisiera además, avanzar en la calidad visual de las páginas presentadas, se deben definir nuevos controles web de usuario que cumplan con las expectativas del caso. A fin de poder ser introducidos de forma simple al entorno del Workflow, cuando un usuario administrador defina un esquema, puede asociarle a cada campo, un nombre de control con el que se vea en la grilla y otro con el que se visualice en el formulario. De no definir ninguno en particular, se seleccionará el más adecuado para el tipo de datos de la columna. Cada control tiene una serie de propiedades a definir y las más importantes son:

- **Etiqueta:** Propiedad que establece para cada campo, la etiqueta de la que vendrá precedido en el formulario.
- **Valor:** dependiendo del modo en el que se despliega el formulario {Insert, Update, Display}, puede o no tener el campo un valor asociado. Obviamente en modo Insert no tendrá valor asociado previamente, pero en los otros puede tenerlo. El valor se saca del valor de la columna adecuada, en el registro a partir del cual generamos el formulario.
- **ToolTip:** Es otra de las propiedades que tiene todo control web de usuario. El ToolTip es el texto emergente que se despliega, generalmente sobre fondo crema, al posicionar el puntero del mouse sobre un control. Este texto, será asociado por el usuario administrador, cuando define el campo y el editor de formularios, solo lo setea al control adecuado. El hecho de que éste se despliegue es propiedad de los controles de usuario que, si el valor de esta propiedad no es nulo, lo muestran cuando corresponda.
- **WidthPorcent:** Es el porcentaje del ancho de la celda asociada al control que ocupará el mismo. En un paso previo al de agregar cada control al formulario, se recorren las columnas, obteniendo el largo máximo de definición del campo. En esta propiedad, se define el porcentaje del ancho, a partir del ancho propio y el máximo.
- **ReadOnly:** Indicador de solo lectura. Un campo, por si mismo tiene la propiedad de solo lectura, que puede o no estar activada. Aparte, el formulario como tal, puede ser también de solo lectura. Por tanto, al campo se le activa la propiedad de solo lectura si alguna de las dos (la del formulario o la suya propia) está activa.
- **Cantidad máxima de caracteres que almacenará el campo.**
- **Largo Decimal:** Si el tipo de datos de la columna es decimal, se consulta por la propiedad de largo decimal. Si está establecida, se pasa al control su valor. Si no está establecida no se define y, por tanto, soporta cualquier cantidad de decimales.
- **AllowNulls:** Indicador booleano de si debe o no permitir el ingreso de valores nulos al procesar los datos ingresados.
- **Método SetValidationMode:** Este método expuesto por los controles, por medio de la implementación de la interfaz IICControl hace que la forma de definir a un control su método de validación quede estandarizada.

SchemaWeb_Main.

SchemaWeb_Main es el control de usuario que engloba la lógica necesaria para presentar en pantalla un formulario adecuado para visualizar el contenido de un esquema. Como se ha manejado ya en varias oportunidades, un esquema es un DataSet con algunas propiedades extra, que se definen mediante el SchemaDefinition.

El SchemaWeb_Main, tiene por objetivo presentar un formulario web que represente gráficamente la estructura de un esquema. Un esquema se conforma de tablas (compuestas de columnas), registros dentro de las tablas, y relaciones entre campos de distintas tablas. Por tanto, el presentar la estructura del esquema en un formulario, puede ser simplificado a interpretar las relaciones entre tablas y delegar la tarea de presentar el contenido de tablas y registros a terceros controles. Éstos, los controles que resuelvan la presentación de las tablas (SchemaWeb_Grid) y registros (SchemaWeb_Form), serán analizados más adelante en esta misma sección, pero básicamente se puede reducir su tarea, a presentar adecuadamente los datos de una tabla y de un registro respectivamente. Por el resto de esta sección se asume esta idea, para presentar el control SchemaWeb_Main.

La idea es obtener un formulario con el estilo que se presenta a continuación:

idFact	cantidad	descripcion	TotalFact
1166	Recursos de sistema	Suspendido	00/00/0000
1167	Recursos de sistema	Suspendido	00/00/0000
1168	Recursos de sistema	Suspendido	00/00/0000
1169	Recursos de sistema	Suspendido	00/00/0000
1170	Recursos de sistema	Suspendido	00/00/0000
1171	Recursos de sistema	Suspendido	00/00/0000
1172	Recursos de sistema	Suspendido	00/00/0000
1173	Recursos de sistema	Suspendido	00/00/0000

Formulario dinámico en tiempo de ejecución

Para lograr este formulario, el SchemaWeb_Main, tiene una serie de pre-requisitos, que debe satisfacer el proceso que lo llama. Satisfechos éstos, será capaz de desplegar un formulario adecuado. A continuación, se presentan la instanciación de un objeto SchemaWeb_Main, la forma en que se genera el formulario, se le carga información y se guardan en el esquema los cambios.

Instanciación:

Al llamar a una página web, que presente un formulario adecuado para desplegar la información contenida en una instancia de un esquema, debe establecerse en la llamada a la misma, el número de tabla y registro a partir del cual comienza a editarse el formulario.

Otro parámetro importante es el DataSet a partir del cual debe generarse el formulario. El control presenta una propiedad pública llamada justamente DataSet, que permite setear y obtener este valor. El SchemaWeb_Main a partir del DataSet, la tabla y registro por la que debe comenzar la presentación, comienza a interpretar las relaciones entre tablas y generar la estructura del formulario.

En definitiva, son cuatro parámetros los directamente relacionados con la estructura del esquema, su tratamiento y la información a presentar:

- **DataSet:** Propiedad pública de la clase que permite acceder y modificar el DataSet, a partir del cual se infiere el esquema para el que se desplegará el formulario.
- **InitialData:** Información a desplegar en el formulario que se presente. Es un string conteniendo la serialización de un archivo XML, con los datos.

- **TableIndex:** Índice de la tabla principal, es decir, aquella a partir de la cual se comienza la navegación por las relaciones existentes en el esquema. En el ejemplo ya manejado de la factura, sería el cabecal.
- **RowIndex:** Índice del registro de la primera tabla a partir del cual se comienza la edición del formulario para el esquema.

Hay una serie de propiedades extra que quien llama al SchemaWeb_Main debe setear que están más orientadas a la visualización general del formulario, independientemente del esquema que se presente. Ellas son:

- Propiedades utilizadas para propagar parámetros a las grillas contenidas en el formulario: Son indicadores booleanos de si se debe o no presentar determinados botones que permiten al usuario desarrollar distintas acciones. Según el caso del esquema que se desee presentar, se modificarán los valores adecuadamente por parte del objeto que llame a SchemaWeb_Main, pero en principio se inicializan todos en verdadero. Luego al generar el formulario, los distintos botones se agregan o no, de acuerdo al valor de estas propiedades, por lo que si no se hace nada particular con ellas, estarán en verdadero y se presentarán todos los botones.
 - ShowInsertButton: Permite definirle a las grillas contenidas en el formulario si presentar o no botones de inserción de nuevos registros.
 - ShowPrintButton: Al presentar una grilla puede o no hacerse con un botón de impresión en la parte superior. Esta tarea la resuelve el control específico de las grillas, pero el indicador es propagado a todas ellas desde el formulario principal.
 - ShowDisplayButton: Al presentar una grilla, por cada registro se podrá agregar un botón de display. La idea es que al presionarlo se presente el contenido del registro en un formulario de solo lectura. El indicador de si este botón estará o no en la grilla, es publicado a nivel del formulario principal.
 - ShowUpdateButton: De forma similar al anterior ocurre con un botón que permite editar y modificar registros en la grilla. Se presenta el contenido en un formulario, pero en este caso no será de solo lectura.
 - ShowDeleteButton: Indica si se agrega un columna más por registro, que al ser presionado lo borre de la grilla. Al igual que antes esta propiedad se establece y publica a nivel de todo el formulario, para luego ser propagada a cada grilla particular.
- ShowAcceptButton: Indica si al final del formulario se agrega un botón de confirmación de cambios. En caso que este indicador sea positivo y el botón se agregue, al ser presionado por un usuario debería disparar un evento que impacte en el DataSet los cambios realizados sobre el formulario.
- ShowCancelButton: De forma complementaria al indicador anterior, éste permite definir si se presenta un botón de cancelar.
- ReadOnly: Indicador booleano utilizado a la hora de generar el formulario para saber si la información presentada puede o no ser modificada por el usuario. Este indicador debe propagarse a todas las grillas y todos los formularios que compongan el esquema.

2.7 Generación del Formulario.

El primer paso es obtener la tabla indicada por el índice "TableIndex". Ésta será la tabla principal del esquema, a partir de la cual, se generará todo el formulario. Acto seguido, se debe obtener el indicador de visibilidad y la cardinalidad de la misma. En caso de ser una tabla no visible, no se despliega nada, dado que en realidad no se debe nunca llegar hasta este punto, porque no tiene sentido que la tabla principal del esquema sea invisible para los usuarios. Para las tablas visibles, se despliega el formulario adecuado.

Lo primero que se decide es el formato general del formulario. Existen dos formatos básicos, entre los que se discrimina utilizando el índice del registro a editar, como bandera. Si este último es -1, es un indicador de que la intención es ingresar un registro nuevo, no modificar uno existente. Por otro lado, si la necesidad es presentar un determinado registro el caso es diferente, ya que no estaríamos ante una inserción sino una modificación de información existente. En este caso la propiedad RowIndex tendrá un valor válido cargado (entre cero y la cantidad de registros de la tabla base, menos 1). En resumen, las opciones son:

- **Formulario en Modo Insert:** El indicador de registro activo es -1 (valor inválido utilizado como bandera). Se debe presentar un formulario vacío, con todas las estructuras necesarias (subformularios y grillas) para insertar una nueva instancia del esquema representado por el DataSet.
- **Formulario en Modo Update o Display:** El RowIndex es mayor o igual a cero. En este caso, se presenta el formulario similar al anterior, pero con información cargada previamente.

Las anteriores opciones se detallan más adelante, pero lo que hacen es cargar un arreglo local de controles de usuario, con los controles (formularios y grillas) necesarios para presentar el esquema de forma amigable.

El último paso, luego de tener los controles ya generados es agregarlos a la página web, de forma inteligente. Lo que se ha decidido hacer en este caso es recorrer el arreglo de controles y en cada caso llamar al método privado `AddControlToPage`, pasando cada uno de ellos.

Formulario en Modo Insert.

Obtener un nuevo registro de la tabla (en blanco). A partir de él, obtener el formulario. Obtener todas las relaciones de la tabla, donde es tabla madre, es decir, todas las que apunto yo, son hijas mías, o como sea. Para cada relación busco la tabla hija. Si debe ser visible, pregunto por la cardinalidad. Si es 1, esta tabla tendrá un único registro, por lo que será presentada en forma de formulario. Llamar al método que obtiene el formulario (`GetForm`), pasando un registro en blanco de la tabla y un registro en blanco de la tabla madre. Si la cardinalidad es distinta de uno, se presentará una grilla donde el usuario pueda ingresar todos los registros. Por tanto, se llama al método `GetGrid`, pasando un objeto de la clase `DataRowView`. La `DataRowView` la obtenemos llamando al método `getDataRowViewResult`. Éste nos devuelve una vista por defecto de la tabla a desplegar, si el índice del registro es < 0 y una `DataRowView`, con los registros hijos del registro indicado de la tabla madre, si el índice es válido (≥ 0). Para hacerlo se vale de la relación (objeto `DataRelation` de ADO.NET) y la propiedad `GetChildRows`, que devuelve un arreglo de registros hijos. Luego crea un clon de la tabla hija y recorre todos los registros en el arreglo, para pasar los datos al clon en blanco. Por último crea una vista sobre el clon cargando los registros hijos, obtenidos anteriormente.

Formulario en Modo Display o Update.

En este caso, claramente la intención es presentar la información referente al registro indicado por `RowIndex` y sus posibles registros hijos en otras tablas. El paso primario consiste en generar la estructura básica del formulario para visualizar los datos del registro indicado de la tabla base. El siguiente paso es exactamente igual a lo que se hace en el caso de ingresar un nuevo registro. Se recorren todas las relaciones en las que la tabla base es tabla madre y, para cada una de ellas se obtiene la tabla hija. En caso de que ésta sea visible, se presentará un control adecuado, donde puedan ingresarse los datos. Nuevamente a partir de la cardinalidad de la tabla, se decide si se utiliza un control de tipo formulario o grilla. Si la cardinalidad es 1, se utiliza un formulario, por lo que se llama al método `GetForm` pasándole el único registro de la tabla hija, como registro a presentar y el registro indicado por `RowIndex` en la tabla madre, como registro padre de la relación en el formulario. Si la cardinalidad es distinta de uno, se presenta una grilla, llamando al método `GetGrid` con los mismos parámetros que en el caso 1, es decir, llamando al método `GetDataSetView`, pasándole por parámetro el DataSet, la relación evaluada, el índice de la tabla y el índice de la fila.

En cualquiera de los casos anteriores, con distintos parámetros, se llama siempre a los métodos `GetForm` (obtiene un formulario adecuado para presentar un registro en pantalla) y `GetGrid` (obtiene una grilla para presentar los datos de una tabla). A continuación se analiza cada uno de estos métodos.

Método GetForm.

El método GetForm encapsula toda la llamada y generación de un control de usuario adecuado para la presentación de una DataRow. DataRow es uno de los objetos más importantes del modelo de objetos del DataSet en particular y ADO.NET en general. Es la representación de las filas de una tabla en el modelo relacional clásico.

```
Private SchemaWeb_Form GetForm(DataRow pDataRow,  
DataRowpParentDataRow)
```

Cabecal del método, definido en C#

Recibe la fila de datos a presentar y la fila padre de la relación que ha generado este formulario. Básicamente se puede resumir su tarea a cargar una instancia del formulario SchemaWeb_Form, setearle las principales propiedades y devolverlo.

Para cargar el control de usuario se utiliza el método "LoadControl" de la clase base UserControl, asignando el resultado a un objeto local del tipo SchemaWeb_Form. Las propiedades básicas que se establecen desde el main al formulario resultante son:

- ControlID: Identificador de control que tienen todos los controles de un formulario, por el hecho de heredar de SchemaWebControl. El identificador se forma con el idÚnico del formulario que se está creando, seguido de un contador de controles.
- DataRow: Le setea al control de formulario la DataRow que debe desplegar.
- ParentDataRow: Le setea la DataRow padre en la relación que ha generado la necesidad de presentar el formulario que se esté creando.
- ReadOnly: El esquema main, puede ser presentado en formato ReadOnly, es decir, de solo lectura. En este caso, dicha propiedad debe propagarse a todos los controles de grilla y formulario que compongan el SchemaWeb_Main.

Método GetGrid:

El método GetGrid encapsula la llamada y generación del control de usuario adecuado para la presentación de una DataView. El objeto DataView pertenece a las bibliotecas de ADO.NET, de .NET Framework y es una vista de los datos de una tabla de datos. La vista se puede obtener copiando la estructura de la tabla y filtrando los datos por una determinada condición. Hay un método de la tabla (DataTable::GetDefaultView), que devuelve una vista por defecto, es decir, con todos los registros, a partir de la tabla y sus datos.

```
private SchemaWeb_Grid GetGrid(DataView pDataView)
```

Cabecal del método, definido en C#

Recibe la vista de datos a presentar. Básicamente se puede resumir su tarea a cargar una instancia del control SchemaWeb_Grid, setearle las principales propiedades, agregarle los controles de acción que correspondan y devolverlo.

Para cargar el control de usuario se utiliza, al igual que para todos los objetos derivados de SchemaWeb_Control, que hereda de UserControl, el método "LoadControl", asignando el resultado a un objeto local del tipo SchemaWeb_Grid. Las propiedades básicas que se establecen desde el main a la grilla resultante son:

- ControlID: Identificador único del control.
- DataView: Vista a partir de la cual se debe generar la grilla.
- TableIndex: Índice de la tabla en el DataSet.
- ShowInsertButton: Indicador de si debe o no, visualizarse el botón de inserción de nuevos registros en la parte superior de la grilla que se genere. El botón siempre esta presente, simplemente lo habilita o deshabilita según este parámetro.
- InsertUrl: URL de la página que se desplegará para inserción de registros en la grilla.

- ShowPrintButton: Indicador sobre la intención o no de presentar un botón de impresión. El evento asociado al botón invocará a un JavaScript que, por ahora, imprime todo el contenido de la página. En un siguiente paso, se podría cambiar el código del script para que haga algo más interesante, pero no es el objetivo en esta versión del producto.
- ReadOnly: El esquema main, puede ser presentado en formato ReadOnly, es decir, de solo lectura. En este caso, dicha propiedad debe propagarse a todos los controles de grilla y formulario que compongan el SchemaWeb_Main.

Además de establecer el valor deseado para todas las propiedades que rigen la generación de la grilla, existen tres controles más que se presentarán o no en la grilla, de acuerdo al valor de propiedades en el esquema principal que se transfieren a la grilla. La grilla publica una colección de controles, llamada ActionControls. ActionControl es un arreglo de objetos de la clase SchemaWebGridActionControl, que se presenta junto a la grilla en la siguiente sección. Lo que hace el main es, valiéndose de los valores de sus propiedades booleanas ShowDisplayButton, ShowUpdateButton y ShowDeleteButton, agrega o no elementos al arreglo ActionControls. Los elementos del arreglo serán luego utilizados por la grilla, para agregar a cada registro botones de:

- Display: Muestra el registro en un formulario, en modo solo lectura.
- Update: Muestra el registro en un formulario, pero permite modificaciones:
- Delete: Elimina el registro de la tabla.

El funcionamiento de los ActionControls se estudia en la presentación de la grilla (SchemaWeb_Grid).

2.8 Cargar Datos en el Formulario.

La carga de datos se compone de dos instancias básicas. La primera, es adjuntar al DataSet que se utiliza como base de toda generación del formulario, el contenido del XML con los datos que debe setear en la propiedad InitialData quien llama al SchemaWeb_Main. Esto es muy simple, si se hace utilizando la potencialidad de .NET para manejar XML y DataSets. A continuación se presentan las tres líneas necesarias para hacerlo.

```
private void LoadData(DataSet pDataSet, string pXMLData)
{
    StringReader IData = new System.IO.StringReader(pXMLData);
    pDataSet.ReadXml(IData);
    pDataSet.AcceptChanges();
}
```

Cargar el DataSet a partir de un string (XML)

La sentencia AcceptChanges del DataSet, es un método que brinda la clase DataSet e inicializa todas las banderas en sus valores por defecto. Las banderas en las que el DataSet basa su labor están asociadas en general a los registros de cada tabla e indican si un registro es nuevo, ha sido modificado, ha sido eliminado, etc. Básicamente resetea todas esas banderas, para que luego, los cambios que realice el usuario, vayan modificando dichas banderas. A la vez, se puede adelantar que estas banderas serán las utilizadas por la herramienta de actualización de cambios en el DataSet, a la fuente de datos (SchemaImpact, presentada posteriormente).

Esta primera instancia deja todo preparado para la segunda instancia de carga de datos en el DataSet. La segunda se da en el evento Load de cada control, donde se obtiene de la DataRow en el caso del formulario y de la DataView en el caso de la grilla los datos respectivos. Obviamente tanto la DataRow, como la DataView, se obtienen del DataSet, por lo que tienen los datos cargados en la primera instancia a partir del XML.

Guardar cambios en el esquema.

Hasta ahora hemos analizado como se genera el formulario a partir de un esquema y como se carga con los datos, a partir de un documento XML, si es que lo hay. Una vez presentado al usuario, éste hará modificaciones a la información contenida en el formulario. Cuando considere que ha terminado, deberá presionar un botón de aceptar (como todo formulario clásico con un botón de aceptar y uno de cancelar). En el control `SchemaWeb_Main`, está asociada al evento que genera el click sobre el botón, la llamada al método privado `SaveData`. La idea primaria es que cada control que compone el `SchemaWeb_Main` debe conocer la forma de impactar en el `DataSet` los cambios realizados por el usuario, sepa como impactar sobre el `DataSet` de base los cambios que el usuario realiza en su información. Por tanto, el `SchemaWeb_Main`, debe ser capaz de guiar el impacto de la información por parte de los controles que lo componen. Los controles incluidos en el main serán grillas o formularios. Cada uno de ellos, implementa un método `SaveData`, a partir de un `DataRow` recibido por parámetro (el registro padre de la relación entre el main y el control actual). El main, obtiene los controles que lo componen a partir de la session y recorre la lista. En cada caso, verifica si el control que está por impactar, es el formulario de la tabla base. En este caso, llama al método `SaveData` del control, pasando como `ParentDataRow` el valor nulo. Esto se debe a que la tabla base no tiene registro padre. El resto de los controles, sean grillas o formularios serán derivados del registro objetivo de la tabla base. Por tanto en la invocación al método `SaveData`, se les debe pasar por parámetro el registro con los datos del objetivo en la tabla base.

SchemaWeb_Grid

El SchemaWeb_Grid es un control web de usuario que tiene por objetivo fundamental el posibilitar la presentación de una grilla, de forma amigable para el usuario, a partir de una vista de datos (DataView de ADO.NET). Un objeto de la clase DataView es el símil en el modelo de objetos de ADO.NET a lo que es una vista en base de datos. En el caso particular de esta aplicación se utiliza de forma análoga a lo que sería una tabla de datos (DataTable). Se define en base a una DataView, porque ésta brinda algunas funcionalidades que la tabla no, por extensibilidad para aprovecharlas luego, si surge la oportunidad. Los posibles objetivos son presentar una grilla vacía, una con todos los datos de una determinada tabla, o con parte de ellos. Para los dos primeros casos, se puede utilizar el método "DefaultView", que devuelve el objeto DataView que responde a la vista por defecto sobre la tabla. A partir de la DataView, se construirá la grilla adecuada. Para el tercer caso, se recorren los registros de la tabla, verificando por alguna condición y los que la cumplan serán agregados a la vista. El objetivo es llegar a una grilla que presente los datos, como vemos a continuación.



ID	Proceso	Estado	Hito	Apertura
1188	Recepción de reclamos	Suspendido	Escalado	01/10/2003
1183	Recepción de reclamos	Suspendido	Escalado	30/09/2003
1181	Recepción de reclamos	En ejecución	Ingresado	30/09/2003
1173	Recepción de reclamos	Suspendido	Escalado	29/09/2003
1154	Recepción de reclamos	Suspendido	Escalado	25/09/2003
1153	Recepción de reclamos	Suspendido	Escalado	25/09/2003
1142	Recepción de reclamos	Suspendido	Escalado	25/09/2003
1141	Recepción de reclamos	Suspendido	Escalado	25/09/2003
1116	Recepción de reclamos	Suspendido	Asignado	22/09/2003
1114	Recepción de reclamos	Suspendido	Asignado	22/09/2003

Una grilla en tiempo de ejecución

En esta sección se analiza principalmente la instanciación de la grilla, la generación automática de la misma a partir de la vista de datos, como se presentan los datos en pantalla, y como se agregan los botones de acción necesarios para generar una navegabilidad adecuada a través del esquema, así como permitir la modificación de los datos.

Instanciación de la grilla.

El Editor de Formularios a partir de un esquema, clase SchemaWeb_Main, cuando presenta una tabla lo hace en forma de grilla. Para esto, el primer paso es cargar el control de usuario SchemaWeb_Grid, y setearle sus propiedades básicas. Las mismas son:

- ShowInsertButton: Indicador booleano de si debe visualizarse o no el botón de insert. En caso de permitirse el mismo, en el evento asociado al botón se levantará un formulario para el ingreso de los datos.
- InsertURL: URL asociada al linkButton "Nuevo". Es el que presionará el usuario cuando pretenda agregar registros a la grilla. Si en la propiedad anterior se ha seteado verdadero, en esta debe ingresarse el link de la página que presentará el formulario para el ingreso.
- ShowPrintButton: Indicador booleano de si debe o no mostrarse el botón de imprimir.
- TableIndex: Índice de la tabla del esquema que se desplegará en la grilla.
- ControlID: Identificador único del control. Se crea a partir del control que lo contiene y agregando el contador de controles dentro de éste.
- DataView: Es la vista a partir de la que se crea la grilla.
- ReadOnly: Indicador de solo lectura. Una grilla será de solo lectura, si la tabla que se desea presentar en ella lo es o si se está generando un formulario de solo lectura, en cuyo caso este indicador se replica en cascada a todos los controles en la página.

Además de las propiedades anteriores, en el proceso de instanciación, el Main le agrega a la grilla los botones de Display, Insert y Update. Dependiendo del modo en el que se abra el formulario principal, éste puede o no tener establecidas las propiedades ShowDisplayButton, ShowInsertButton, ShowDeleteButton en verdadero. En la etapa de instanciación de una grilla, el main, para cada una de estas propiedades que esté habilitada, agrega un botón para satisfacer dicha acción de control. Luego le setea a la grilla, un arreglo de controles, con los que haya considerado necesarios.

Con todas estas propiedades seteadas, el control es agregado por el main a la página, por lo que previamente a levantarse la página en el cliente, se arma la estructura de la misma y se ejecutan en cascada los métodos Page_Load de cada uno de los controles involucrados. En el caso de SchemaWeb_Grid, el método Page_Load tiene el cometido de armar en pantalla la grilla adecuada para visualizar la vista que se recibe por parámetro, a partir de un objeto DataGrid, miembro de la clase.

DataGrid: El objeto DataGrid es la representación en ASP.NET de la grilla de datos. Básicamente, se puede decir que expone todas las funcionalidades necesarias para administrar una grilla. Permite agregar o eliminar columnas, cambiar títulos, ordenar los registros, agregar registros nuevos, modificarlos por parte del usuario, etc.

Armado de la grilla.

El evento Page_Load, tiene el objetivo de armar toda la estructura visual del control para que éste pueda ser presentado adecuadamente, basado en la estructura de la tabla (las columnas) y las propiedades que le setea el SchemaWeb_Main.

Por un lado, debe resolver cuestiones anexas, como ser, visualización o no de la etiqueta de descripción de la grilla (en función de que esté o no definida la propiedad en la tabla que quiero visualizar), mostrar u ocultar los botones de impresión y nuevo registro, según el valor de las propiedades seteadas por el main, ocultar todo el panel del cabezal si no va a mostrar ni la descripción, ni los botones, etc.

Por otro lado, debe resolver la estructura de la grilla propiamente dicha. En este caso hay dos propiedades importantísimas, para resolver el problema. Ellas son:

- Visible en grilla: Indicador booleano de si el campo debe o no ser mostrado en la grilla.
- Índice en grilla: Índice con el cual debe ser visualizado en campo en la grilla.

En el momento de guardar los índices, al crear los esquemas, se agrega una columna al mismo (invisible en formulario, invisible en grilla, no modificable, y demás características para que no influya en la utilización estándar del esquema), sobre la que se define una restricción de unicidad. Esta columna se llama GridIndex. Lo que hace el visualizador de grillas, conociendo la existencia de esta columna es ordenar los registros, por el valor de esta propiedad. Una vez ordenados, se recorren todos, verificando el valor de la propiedad "visible en grilla". Todas las columnas que tengan esta propiedad activada, son agregadas a la grilla en pantalla.

Una vez agregadas todas las columnas, se recorre el arreglo de objetos SchemaWebGridActionControl que posee la grilla. SchemaWebGridActionControl es un objeto privado del proyecto y tiene tres campos:

- El nombre.
- El tipo de control.
- El índice en el cual se despliega en la grilla.

Esta colección ha sido seteada por el main y puede tener cada uno de los siguientes tres SchemaWebGridActionControl:

- ShowDisplayButton: Botón de visualización.
- ShowUpdateButton: Botón de Insert
- ShowDeleteButton: Botón de Delete.

Cada uno de ellos, se presenta como una columna más en la grilla. Las columnas que se agregan son objetos del tipo `HyperLinkColumn`, lo que permite asociarles una imagen y un link. Por tanto, se agrega una imagen adecuada y un link dependiente de la columna (tarea resuelta por quien instancie la grilla, en el caso general será el `SchemaWeb_Main`). Ese link redirige a una página donde se presentan los datos del registro seleccionado (mediante el editor del formulario), en formato de solo lectura se selecciona "Display", en formato de modificación, si seleccionamos "Úpate" y a una página que elimina el registro marcado y vuelve automáticamente a la actual, si se selecciona "Delate".

SchemaWeb_Form.

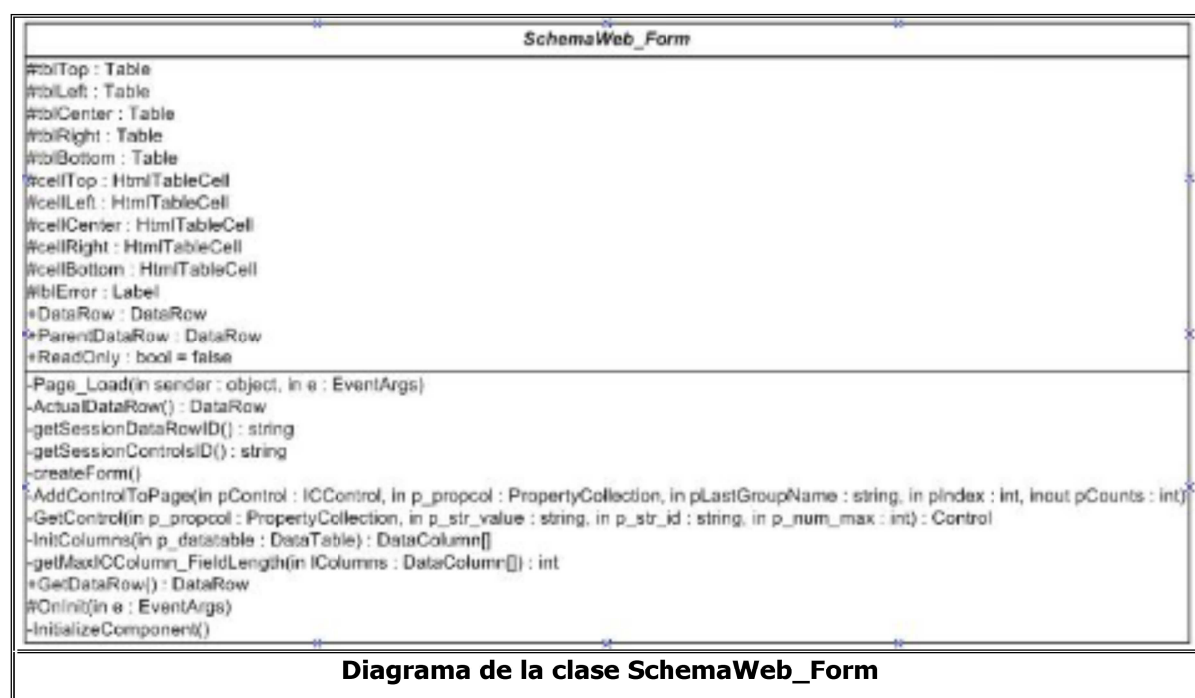
El SchemaWeb_Form es un control de usuario (se crea heredando de la clase UserControl de ASP.NET). En el proyecto es utilizado al desplegar el esquema por el control SchemaWeb_Main, que cuando decide que debe desplegar un formulario, crea una instancia de este control y la agrega a su colección de controles.

Instanciación:

El main debe resolver que registro desea desplegar y cual es el registro padre. Por registro padre, se denomina al registro que contiene la clave primaria de una clave foránea del registro objetivo. En el ejemplo de la factura, al editar el registro de la tabla cabecal, no se tiene registro padre, pero al generar un formulario para los datos de cada línea, el registro padre es el cabecal. Esto lo resuelve el SchemaWeb_Main, a partir de las relaciones en el esquema (dado que existe una relación entre las tablas). Una vez que conoce ambos registros (el objetivo y el padre, si éste existe), carga el control (SchemaWeb_Form) valiéndose del método de la clase base UserControl LoadControl, a partir del nombre. Acto seguido, debe establecerle el identificador único (lo crea a partir del control main, más un numerador único), el registro objetivo, el registro padre y un identificador de "Solo Lectura". Al crear el formulario principal, se establece esta propiedad y luego éste la propaga a todos los controles que lo componen. El siguiente paso, para la instanciación de un control del tipo SchemaWeb_Form es agregarlo a la lista de controles. Al momento de desplegar la página, se mostrarán todos los UserControl de la colección, a la vez que se disparan en cascada de sus eventos Load.

Diseño.

Se presenta en primera instancia el diagrama de la clase, obtenido mediante la utilización de Visio.



Como se puede deducir de los primeros diez atributos de la clase, el formulario básico se conforma de cinco tablas, con una celda cada una. Dentro de estas celdas es que se colocan los controles que dinámicamente se agregan, de acuerdo al esquema particular. Cada campo a agregar al formulario, se dirigirá hacia alguna de esas celdas. La etiqueta de error es donde el formulario mostrará mensajes adecuados, ante particularidades detectadas, que deban ser informadas al usuario. Ejemplos de error son, excepciones capturadas al intentar impactar en el DataSet los cambios del usuario, campos que no cumplan la validación, etc.

Los otros atributos, es decir, DataRow, ParentDataRow y ReadOnly son, el registro objetivo, para el cual se despliega el formulario, el registro padre del que se desea desplegar, y un indicador de solo lectura, que debe ser propagado a todos los controles en el interior del formulario.

De los métodos que se observan en la descripción gráfica de la clase, los más destacables son Page_Load y SaveData.

Page_Load: En todos los casos, en el evento Load de un control de usuario, el objetivo es preparar el mismo para que, formando parte de la página esté disponible para la interacción con el usuario. En el caso particular del SchemaWeb_Form, el objetivo de este método es generar un formulario adecuado para la visualización de la información contenida en un registro de una tabla. El registro, como se ha manejado antes, está almacenado en atributo de la clase llamado DataRow (fila de datos, haciendo alusión a la noción de fila-columna de una tabla en modelos relacionales). Por tanto, en base a sus campos, tipos y propiedades, debe armar un formulario adecuado. Se puede dividir la tarea del evento Page_Load del control formulario en tres etapas que se repiten para cada columna de la DataRow:

- **Elegir el Control más adecuado:** La idea general es recorrer todos los elementos de la colección de columnas de la fila y, para cada una de ellas, buscar la forma de desplegar un control adecuado para presentarlo al usuario. Lo primero que se hace es verificar el valor de la propiedad que indica si el campo debe ser visible en formulario. En caso negativo, no se agrega, pero si debe ser visualizado se sigue adelante con el proceso. A cada columna, a la hora de definir el esquema se le puede asociar un nombre de control para formulario. Este debe ser el nombre de un UserControl definido previamente. El método encargado de cargar el control adecuado para cada columna, verifica el valor de esta propiedad y trata de cargar el control con ese nombre. Si la propiedad no ha sido cargada o su valor es nulo, se fija en el tipo de datos que almacenará el campo. Se han definido una serie de controles de usuario con las funcionalidades requeridas para los tipos de datos más comunes (string, entero, decimal, fecha, fecha y hora, texto largo, boolean, etc.). Como se ha presentado en la sección de la herramienta de definición de esquemas, los tipos de datos que ésta permite definir, son cargados desde un XML de configuración, por lo que agregar un nuevo tipo, se reduce a agregar una línea al XML y, antes de ponerlo en producción, definir el control de usuario que utilizará el Workflow para presentar campos que almacenen información de este tipo. Por último, si ninguna de las opciones anteriores han resultado exitosas, se cargará un control genérico que toma cualquier tipo de información que reciba como String y la guarda.
- **Seteo de Propiedades a cada control:** En esta sección se analiza el seteo de propiedades a los controles de usuario. Cada control brinda la posibilidad de setearle una serie de propiedades. El formulario pregunta por cada una de las propiedades que debe tener el control y las va seteando de a una. Para ver cuales son y que utilidad tienen repasar la definición de los controles de usuario.
- **Agregarlo al formulario:** Una vez definido el control que se utilizará para el campo, debe ser éste agregado a la colección de controles del formulario. Para agregarlo hay una serie de decisiones a tomar, nuevamente basadas en propiedades del campo. El formulario, como tal, consta de cinco tablas (superior, izquierda, centro, derecha, inferior), con una celda cada una. Cada columna puede o no tener una propiedad, que indica en que sector se mostrará. Si la tiene instanciada, el editor de formulario agregará el control al sector deseado. Por defecto, en caso de no estar definida o no poder ser correctamente identificada (por extensibilidad y compatibilidad hacia atrás) se agrega en la zona central del formulario. Establecido el sector, se crea una nueva fila de tabla web (objeto TableRow de ASP.NET), dentro de la que se cargan dos celdas. Una de ellas tendrá la etiqueta y la otra el campo para ingresar el valor. Debajo de ésta, se crea otra TableRow con una celda en blanco, que hace las veces de separador. Para simular el separador, se le asocia a la celda una imagen de fondo, que es una pequeña línea gris tenue.

SaveData:

El método SaveData, debe ser implementado por toda clase que implemente la interfaz ISchemaWebControl, siendo este el caso del editor automático de formularios SchemaWeb_Form. Este método debe saber actualizar sobre el DataSet la información modificada por el usuario. En el caso del formulario consiste en tomar los valores ingresados por el usuario e identificando el registro por la clave, pasar los cambios al DataSet. Hay que tener un cuidado extra, con el tema de si el usuario modificó la clave del registro, porque en ese caso, no debemos realizar un insert, sino un update. Para tomar este tipo de decisiones es trascendente el modo en el que se ha editado el formulario.

2.9 Impactador en Base de Datos.

El objetivo de lo que hemos llamado "Impactador en base de datos" es tomar un DataSet cargado con información y, sabiendo que ha sido definido de forma que se ajuste al esquema manejado por nuestro Workflow, guardar esa información en la respectiva base de datos.

La idea, ya manejada, es que al enfrentar una "ParticipantTask", es decir, una tarea que suponga interacción con usuarios (que son parte de los Participants), se despliegue un formulario web y, como vimos en la parte anterior, se brinde la posibilidad de editar los datos y luego de guardarlos. Llegado el momento de guardar los datos, el motor de Workflow debe tener quien le resuelva el análisis de qué datos han cambiado, cuáles se han agregado, cuáles eliminado, etc., para luego impactar esos datos en la base. De forma genérica, la definición de un esquema, permite asociarle al mismo, el nombre de procesos que llevarán a cabo el proceso de Inserción, actualización y eliminación de registros en la fuente de datos. Esto puede ser utilizado para casos particulares que, por su complejidad requieran un tratamiento especial, no generalizable o no estándar, pero el motor, en caso de que no se especifique nada en particular sobre estos procesos, por defecto utiliza una herramienta genérica que recibe un DataSet e impacta todos los cambios experimentados por el mismo, desde la última actualización de base de datos que se haya hecho a partir del mismo. Quien se encargará de este análisis e impacto es el "Impactador de Base de Datos". El análisis tiene como primer etapa y fundamental, el hecho de establecer a partir del esquema y las relaciones, el orden adecuado de inserción, actualización y eliminación de registros en la fuente de datos. Por ejemplo, veamos que ante un esquema simple, de los denominados (master – esclavo), el orden correcto es eliminar del esclavo, actualizar en el esclavo, eliminar en el maestro, actualizar en el maestro, insertar en el maestro, insertar en el esclavo. Un orden inadecuado de las operaciones puede provocar, según la combinación de los datos, inconsistencias en la fuente. Por ejemplo, si queremos eliminar primero los registros de la tabla master, para luego eliminar los de la tabla esclava, el DBMS no nos lo permitiría, mientras que si es válido hacerlo en el orden inverso.

A la hora de impactar el contenido de un DataSet en una base de datos, es de mucha utilidad otra de las clases predefinidas en ADO.NET, el DataAdapter o Adaptador de datos. Un adaptador de datos, en general va asociado a una determinada tabla de un conjunto de datos y con una única invocación a un método update, basado en las propiedades del DataSet, la tabla y sus filas, sabe como impactar los cambios en la base de datos. Lo que debemos definirle nosotros al Adaptador de Datos son las respectivas sentencias de inserción, actualización y eliminación de los datos. Una vez definidas estas sentencias para cada uno de los Adaptadores que nos administrarán cada una de nuestras tablas, lo que nos queda es, basados en las relaciones (colección de objetos DataRelation), definir el orden adecuado de actualización de las tablas.

El siguiente y último paso, sería impactar realmente los cambios, utilizando el orden preestablecido, en función de las propiedades del esquema.

Resumen

A lo largo de este apéndice hemos presentado una serie de conceptos relacionados con la administración de la información que viajará a través de los distintos flujos. En una primera instancia se ha introducido de forma breve la problemática de la administración de la información. Conceptos claves en este tema son permitir:

- manejar la información de una forma suficientemente estructurada, como para que se pueda interactuar con procesos externos al Workflow, sean procesos de la empresa, externos a la misma o incluso, servicios web. Un punto relacionado es que la solución planteada debía permitir imponerse tanto en interacción con procesos provistos por la propia empresa como por terceras partes, de forma transparente.

- debe ser suficientemente abierta como para que el tipo de información a administrar, o las limitaciones que ésta imponga, no deriven en una pérdida de funcionalidad y aplicabilidad del producto final.
- Se pretende presentar una solución en la que, al momento de la implantación y puesta en producción en distintas realidades, la mayor parte del trabajo quede en manos de usuarios administradores del WorkFlow, previendo que éstos tengan las herramientas adecuadas para llevar adelante su trabajo de una forma eficiente y amigable. No es una solución factible, de ninguna forma, una que sea dependiente de la realidad de la empresa en la que se pretenda imponer, no pudiendo tampoco darse el lujo de que particularidades previsibles de la misma, echen por tierra la aplicabilidad del producto prototipado.

De forma paralela se ha presentado el concepto de DataSet y sus ventajas en cuanto a la libertad e independencia con la que permite administrare información en la cargada, lo que lo vuelve una potencial herramienta a utilizar en la solución del problema.

En definitiva, por lo expuesto anteriormente se llega a la conclusión de que lo más adecuado es definir que toda la información que viaje a través de un flujo deba seguir un determinado formato preestablecido por el administrador del sistema a la vez que, en cada tarea se puede definir visibilidades parciales sobre la información a presentar. Así nace el concepto de esquema extendido. Básicamente, un DataSet es una estructura relacional que nos permite trabajar desconectados de la fuente de datos, a la vez que sincronizarnos con ella periódicamente. El DataSet como tal, responde a una estructura relacional y trae resuelta la obtención del esquema XSD al cual responde su estructura. Por otro lado, maneja de forma totalmente transparente la información en la cargada, permitiendo presentarla en forma de tablas y columnas (modelo relacional) o trabajarla como documentos XML que respondan a los esquemas anteriormente nombrados. En base a un DataSet es que definimos nuestro concepto de esquema extendido. La extensibilidad del mismo, pasa por la utilización de las ExtendedProperties que el objeto DataSet prevee asociadas al mismo, a sus tablas, columnas y relaciones. Estas propiedades son guardadas en un formato de Hash, por lo que pueden ser accedidas luego, por su clave. Hemos definido una serie de propiedades necesarias para nuestra administración de la información en el esquema que permiten automatizar partes importantes del proceso.

Nuestra solución supone que en una instancia previa a la definición de un flujo, el usuario administrador del sistema defina el esquema que seguirá la información que viajará a través del mismo. En forma paralela puede definir en cada tarea del flujo, una nueva visibilidad sobre esa información, es decir, que si bien el formato de la información se mantendrá a lo largo de todo el flujo, distintos usuarios o procesos, en distintas etapas del mismo, podrán tener o no acceso a parte de la data, según lo haya definido previamente el administrador. La herramienta que permite definir el esquema de la información, como así también las visibilidades de la misma es el ICSchemaDefinition.

Una vez que hemos definido la estructura que seguirá la información a administrar en un flujo, podemos automatizar dos partes trascendentes del mismo, que de otra forma, requerirían un desarrollo personalizado para cada aplicación. Por un lado se puede automatizar la presentación de un formulario web adecuado para la visualización y modificación de la información, a partir del esquema al que la misma responde. A tales efectos se ha desarrollado la herramienta SchemaWeb_Main, que a partir de la estructura del esquema y sus relaciones entre tablas, infiere el formulario adecuado. Este proceso tiene como etapas fundamentales, la generación de una grilla para la presentación de varios registros de una tabla y, por otro lado, la presentación de un sub-formulario para presentar los campos de un único registro a efectos de su modificación. Se han desarrollado dos controles web (SchemaWeb_Grid y SchemaWeb_Form) que resuelven este tema de forma genérica. A su vez, cada uno de estos debe presentar campos particulares, que responden a alguno de los tipos definidos en el esquema. Cada tipo, tiene asociado un control web que lo presenta de una forma adecuada, por lo que la tarea de presentar un campo particular se deriva a estos controles específicos.

La extensibilidad lograda con esta solución es importantísima y se puede adoptar en dos direcciones fundamentales. La primera es agregar tipos de datos, para lo cual solo se debe agregar una línea al

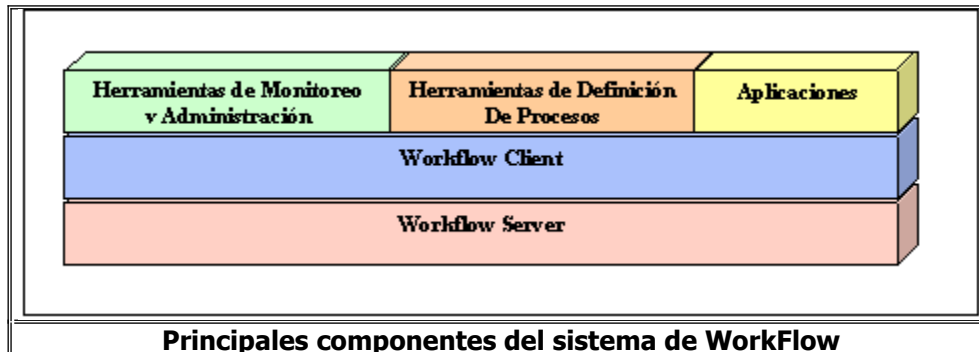
documento XML con los tipos de datos a administrar. Por otro lado, se puede encarar el camino de mejorar la calidad gráfica y las funcionalidades del formulario generado. En este sentido el ICSchemaDefinition, permite definir por cada tabla, un control que la presente en forma de grilla y otro que lo haga en forma de formulario. Si no se define ningún valor válido en estas propiedades se utilizan los controles por defecto, pero si hay un nuevo control desarrollado, se puede utilizar con solo modificar el esquema, en segundos de trabajo del administrador. Por otro lado, también puede el administrador definir un control web específico para presentar cada campo, por lo que se puede mejorar también la presentación de los campos particulares de la misma forma que para toda una tabla o un registro.

El tener definido el esquema que sigue la información, nos permite también automatizar el proceso de impactar la información ingresada en bases de datos externas a la del Workflow. A tales efectos, se ha definido un ToolAgent más, como tantos, que tiene por objetivo hacer este impacto. Por tanto, el administrador cuando está diseñando el flujo, puede seleccionar sitios estratégicos donde impactar el contenido del esquema en la base de datos. Este toolagent se vale de otro proyecto desarrollado en este paquete de administración de información, que hemos llamado SchemaImpact. El SchemaImpact, a partir de un string de conexión a la base de datos y un DataSet con información cargada, actualiza los cambios hechos al DataSet en la fuente de datos. Para hacerlo se vale de las propiedades del DataSet y del DataAdapter. El impacto consta de dos etapas básicas. Una primera etapa infiere del esquema relacional, el orden adecuado en el que impactar los cambios de las distintas tablas. Luego, para cada una de ellas, hace el impacto, valiéndose de las propiedades de visibilidad y modificabilidad por parte del usuario de las distintas tablas y los distintos campos. En forma paralela, a cada esquema se le puede definir, por tabla, un proceso que haga el insert, el update y el delete para actualizar la base de datos. En caso de no estar definidos, por defecto, se utilizará el proceso ya comentado.

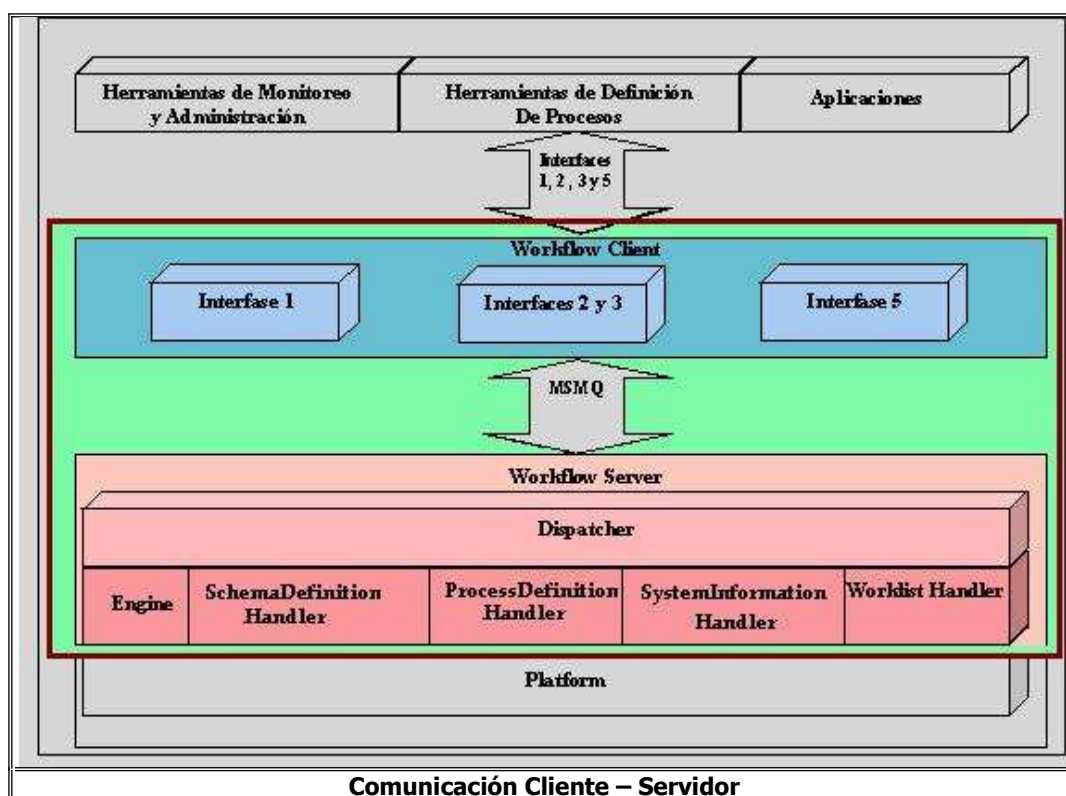
Con la definición del concepto de esquema y estas tres herramientas hemos automatizado buena parte de la administración de la información, a la vez que dejamos la puerta abierta a la evolución hacia mejores soluciones.

3 COMUNICACIÓN CLIENTE – SERVIDOR

Como hemos visto antes, nuestro sistema está formado por una serie de componentes básicos. Entre éstos encontramos, herramientas de monitores y administración, herramientas de definición de procesos, aplicaciones de cliente, el cliente de WorkFlow y servidor de WorkFlow. El siguiente diagrama, presenta de forma gráfica, los anteriores conceptos.



En la presente sección analizaremos el cliente de WorkFlow (WorkFlow Client) y su comunicación con el servidor, por lo que podemos ver con un poco más de detalle en el siguiente diagrama donde se centrará nuestro estudio. Vemos el diagrama anterior, con un poco más de detalle y contrastando en colores nuestro objetivo de estudio, contra una tonalidad gris del resto del sistema, que no nos convoca en este documento.



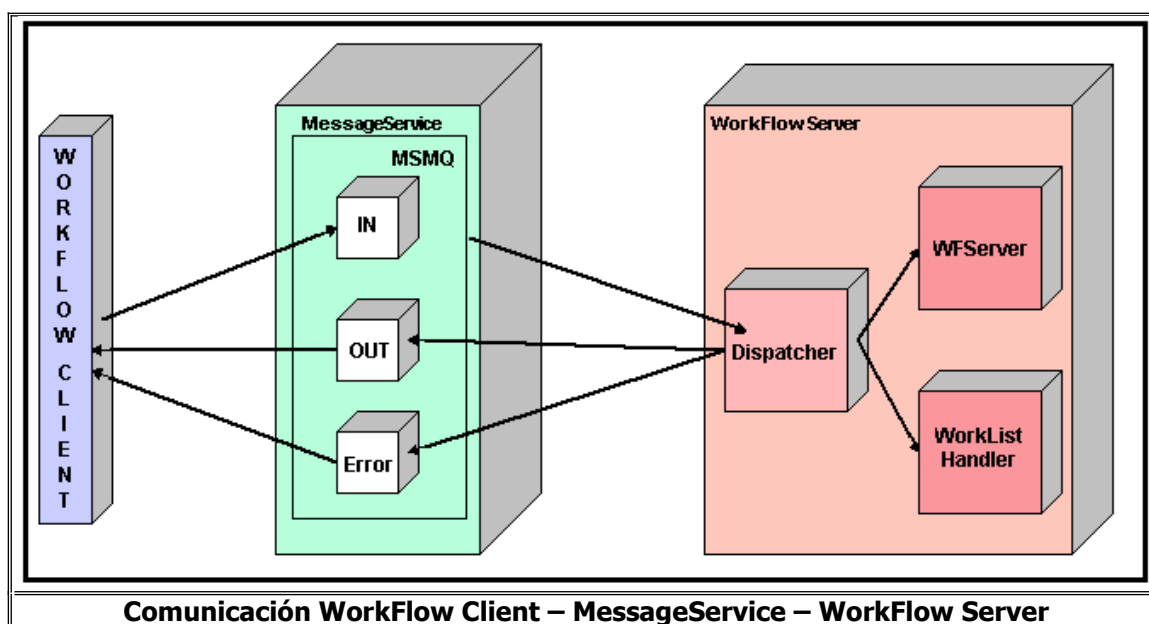
Como vemos, los componentes que estudiaremos son:

- WorkFlow Client (presenta hacia el exterior las interfaces de nuestro sistema).
- WorkFlow Server (Dispatcher, Engine, WorkList Handler).

- Comunicación entre éstos vía Message Queuing. Analizaremos principalmente el servicio de administración de mensajería que utilizamos a fin de administrar el comportamiento de las colas de MSMQ.

Como se observa en los anteriores diagramas, la comunicación entre cliente y servidor se lleva a cabo vía mensajería de Message Queuing. Básicamente Message Queue ofrece un servicio de mensajería en el que definimos colas de mensajes. Una cola, es en definitiva un repositorio al cual varias aplicaciones pueden mandar mensajes y del cual alguna aplicación leerá los mismos. Un mensaje de MSMQ permite definir un identificador, un asunto y un cuerpo de mensaje. A través de estos las aplicaciones pueden comunicarse de forma relativamente transparente. Este es uno de los motivos por los cuales hemos seleccionado MSMQ como mecanismo de comunicación entre el cliente y el servidor, dado que si en un futuro, cualquiera de ellos debe ser modificado, estos cambios no necesariamente impactarán en el otro. Por ejemplo, podríamos fácilmente desarrollar un cliente basado en WebServices, es decir, podemos desarrollar un servicio Web con varios métodos que expongan todas las funcionalidades del Workflow, y comunicarnos detrás de estos servicios con MSMQ, con lo cual ni el Dispatcher, ni el resto del servidor notarían la diferencia.

La idea ha sido definir tres colas de mensajes. Una cola llamada IN será donde el cliente volcará sus mensajes de solicitud al servidor. Éste recogerá los mensajes, actuará en consecuencia y prepondera al cliente por una segunda cola, llamada out. La tercera cola es una cola de ERROR, donde el servidor volcará mensajes con la identificación de las llamadas que han dado un error en el servidor, la descripción del mismo y algún dato más que resulte de interés. En un principio, si bien el servidor volcará mensajes a la cola de error, no se realizará ningún tipo de operación con éstos. La cola se ha definido por extensibilidad, pensando en implementar, en un futuro, operativas de reintento o recuperación al estilo de lo que realiza BizTalk Server, que cuando recibe un error, espera un tiempo y vuelve a reintentar algunas veces antes de notificar el error al usuario. Para que esto sea posible, lo mínimo que necesitamos en el mensaje de error es la llamada realizada, el identificador de la misma y los parámetros para poder reintentar.

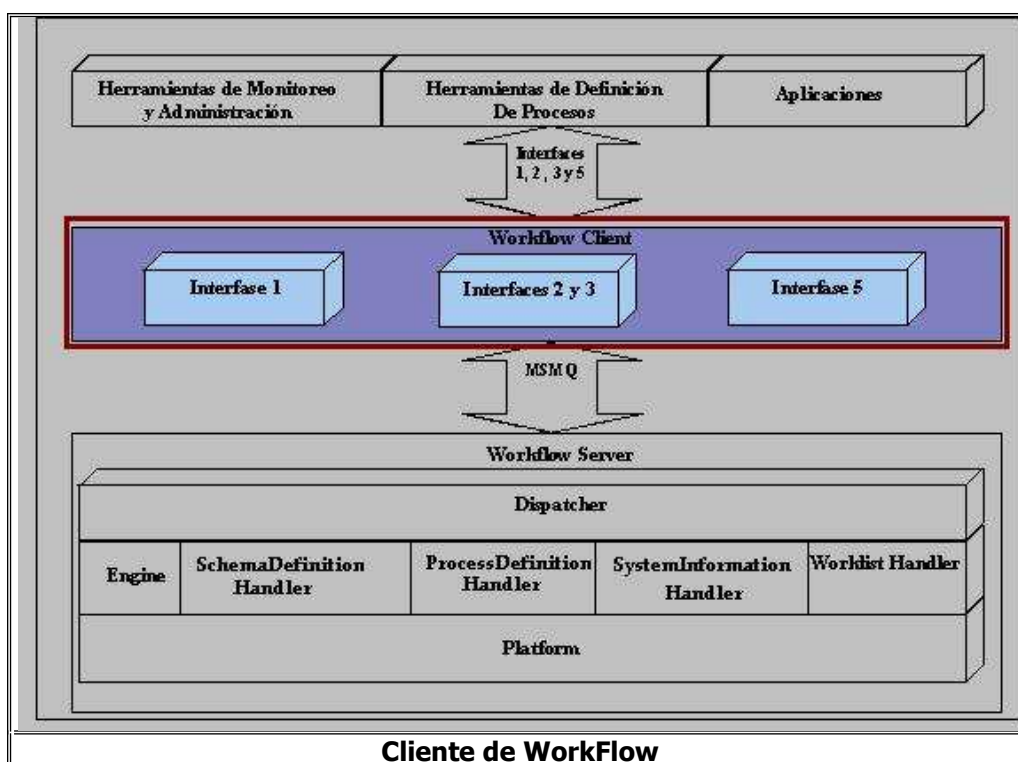


3.1 ICWorkflowClient

La clase ICWorkflowClient, es la cara visible, de nuestro Motor de WorkFlow. Como su nombre lo indica es el cliente, por lo que encapsula todas las llamadas al sistema. Todas las llamadas, sea cual sea el fin de las mismas, que necesiten interactuar con el servidor de WorkFlow, lo harán a través del cliente. Básicamente, el cliente presenta una serie de métodos que exponen todas las funcionalidades que nuestro sistema de WorkFlow brinda. Las mismas se podrían agrupar por funcionalidades en los siguientes subgrupos:

- Conexión y desconexión.
- Administración de definiciones de procesos.
- Administración de la definición de esquemas.
- Administración de instancias de procesos (Process Instances).
- Administración de instancias de actividades (Activity Instances).
- Administración de WorkItems.

Otra posible forma de agruparlas sería por Interfase. Retornando sobre el diagrama original de la arquitectura del sistema de WorkFlow, recordamos que el Cliente expone las funcionalidades propias de las interfaces 1, 2, 3 y 5.



En este caso, agrupadas dentro de la interfaz 1, encontramos a las funcionalidades relacionadas con la definición de procesos, las interfaces dos y tres publican las relacionadas con las aplicaciones de clientes del Workflow y la interfaz cinco publica las funcionalidades expuestas para herramientas de administración y monitores.

- Interfaz 1:
 - Administración de definiciones de procesos.
 - Administración de la definición de esquemas.
- Interfaces 2 y 3:
 - Conexión y desconexión de clientes
 - Administración de instancias de actividades (Activity Instances).
 - Administración de WorkItems.

- Interfaz 5:
 - Administración de instancias de procesos (Process Instances).

Todas esas funcionalidades que expone el cliente, en definitiva publican todas aquellas funcionalidades que el WorkFlow brinda. Por tanto, cada una de estas funcionalidades, se mapea a alguna idéntica que brinda el servidor. El cliente, ante cada solicitud, realiza una llamada al Dispatcher. Como hemos visto en la descripción de la interacción entre el cliente y el servidor, las llamadas se realizan por la vía de intercambio de mensajes en colas de Message Queuing. En definitiva, cualquier operación brindada por el cliente, se traduce en generar el mensaje adecuado y colocarlo en la cola de entrada, de la que lee el Dispatcher. Para colocarlo en las colas, cuenta con un método **privado** llamado CallDispatcher (en alusión a que será el Dispatcher el que luego leerá de esas colas), que realiza dicha tarea.

Conexión y desconexión

Permite conectar y desconectar una determinada sesión de usuario a un determinado sistema. Cuando un usuario se conecta a nuestro motor de WorkFlow, lo hace para interactuar con un determinado sistema. Al hacerlo, de alguna forma el motor le deberá asignar un identificador de sesión a la que está abriendo ese usuario con el motor para trabajar en ese sistema. Este identificador los devuelve como resultado del método de conexión. El resto de las operaciones (todos los métodos expuestos por el cliente de WorkFlow), deben ser llamadas indicando el identificador de la sesión que hace la llamada. Al finalizar la actividad, debería desconectarse del sistema.

Administración de definiciones de procesos

En este conjunto agruparemos a todas los métodos que expondrán las funcionalidades de ABM de definiciones de procesos. El sistema permitirá, como operaciones de administración de sistemas, las siguientes:

- Listar los procesos definidos.
- Obtener la definición de un determinado proceso.
- Agregar la definición de un nuevo proceso.
- Modificar la definición de un proceso existente.
- Eliminar la definición de un proceso.

Administración de la definición de esquemas

La mayor parte de la administración de la información en nuestro WorkFlow se hará a través de lo que hemos llamado Esquemas. Los esquemas son definidos por los usuarios y básicamente brindan todo aquello que puede brindar un DataSet de .NET, dado que se han hecho inspirados en éste, y basados en sus funcionalidades. Como dijimos, los esquemas de administración de la información los definirán los usuarios administradores, por lo el motor debe brindar todas las funcionalidades de ABM de definiciones de esquemas. Las mismas serían, al menos los siguientes:

- Listar los esquemas existentes.
- Obtener la definición de un determinado esquema.
- Agregar la definición de un esquema.
- Modificar la definición de un esquema.
- Eliminar un esquema.

Administración de instancias de procesos

Las instancias de los procesos identifican a las "n" ejecuciones particulares de un determinado proceso P. Por ejemplo, si P es el proceso de aprobación de una orden de compra, en un determinado momento, en el sistema hay tantas **instancias del proceso P**, como aprobaciones de órdenes de compra estén en proceso. Del ejemplo se deduce claramente que sobre una instancia de un proceso hay una serie de operaciones básicas que el sistema debe brindar:

- Listar las instancias de procesos.
- Obtener una instancia en particular.
- Iniciar una determinada instancia.
- Suspender una instancia.
- Detener una instancia.
- Resumir una instancia.
- Reiniciar una instancia.

Administración de instancias de actividades

Como hemos visto antes un proceso consta de una serie de actividades y, por tanto el cliente de WorkFlow debe brindar también una serie de funcionalidades relacionadas con la administración de las instancias de las actividades. Las funcionalidades que debe brindar el cliente son las siguientes:

- Publicar la lista de instancias de actividades.
- Publicar la lista de estados.
- Devolver una determinada instancia, dado su identificador.
- Retomar la ejecución de una instancia.

Administración de WorkItems

Los WorkItems son todas aquellas actividades de un proceso que requieren interacción con usuarios.

Las funcionalidades que brinda el cliente, referentes a la administración de WorkItems son las siguientes:

- Listar las instancias de los WorkItems.
- Listar los estados de las instancias.
- Crear una instancia de un WorkItem
- Contar la cantidad de WorkItems, para una determinada actividad.
- Obtener una determinada instancia de un WorkItem, dado su identificador.
- Detener la ejecución de una instancia de un WorkItem.
- Reasignar una instancia de un WorkItem de un determinado participante a otro.

Comunicación con el servidor:

Nosotros brindamos, a través del cliente, una gran cantidad de funcionalidades (mediante la exposición de distintos métodos). El cliente debe ser capaz de transformar todas esas funcionalidades expuestas en un mensaje de MSMQ, que es el método de comunicación escogido. En tal sentido, encapsularemos el envío de mensajes a colas de MSMQ en un único método, al que hemos llamado CallDispatcher.

CallDispatcher es un método privado del cliente, que encapsula las llamadas al servidor. Ésta debe ser genérica por lo que cada método expuesto debe resolver previamente una serie de problemas como ser:

- Indicarle al servidor claramente el método invocado.
- Asignarle los valores adecuados a los parámetros de la llamada.

Por otro lado, la respuesta a dichas llamadas desde el servidor también serán genéricas por lo que la llamada devolverá en un string la respuesta y cada método debe obtener de forma inteligente a partir de éste, los datos enviados desde el servidor.

- En definitiva, la implementación de cada método expuesto por el cliente, se puede ver como una serie de seis pasos:
 - Indicar la funcionalidad invocada.
 - Resolver el pasaje de parámetros.
 - Armar el mensaje de entrada para el servidor.
 - Colocar el mensaje en la cola de entrada al servidor (IN).
 - Esperar la respuesta por la cola de salida (OUT).
 - Procesar la respuesta.

Determinar la funcionalidad invocada:

La funcionalidad invocada se pasa al método que llama al Dispatcher en un string por parámetro (ejemplos de esto, son "Connect", "Disconnect", "AddProcessDefinition", "DelProcessDefinition", etc.). El servidor sabrá reconocer cada funcionalidad y actuar en consecuencia.

Pasaje de parámetros:

Debemos resolver el pasaje de parámetros sin saber previamente cuántos son, ni mucho menos, sus nombres. En este caso, de ambos lados del canal, debemos conocer los parámetros asociados a cada método. Asumiendo que esto es conocido, el siguiente paso es resolver un método genérico de pasarlos de un lado al otro del canal de comunicación. La solución a este problema, la encontramos por el lado de ingresar todos los parámetros en un hash, utilizando como clave el nombre del mismo.

Armado del mensaje de MSMQ:

Un mensaje de MSMQ tiene básicamente los campos etiqueta, cuerpo y cola a la que se responde. En el campo etiqueta, asignamos el identificador del método. En el cuerpo debemos encontrar la forma de guardar el hash de parámetros. Debemos transformar a string el hash para guardarlo en un mensaje de MSMQ. .NET Framework provee la serialización automática, sin mayor esfuerzo, de un hash a un XML. Por tanto, una vez cargado el hash, podríamos serializarlo del lado del cliente, agregar el XML como parte del cuerpo del mensaje y luego en el servidor deserializar el hash y obtener a partir de éste, el valor de los parámetros consultando el hash por sus nombres.

Colocar el mensaje en la cola de MSMQ:

El siguiente paso es colocar el mensaje en la cola de entrada, de la que lee el servidor. Debemos definir un objeto del tipo MessageQueue, a partir de su nombre y el equipo en el que se encuentra corriendo. Luego formamos un mensaje (objeto de tipo Message), le cargamos los valores y luego a la cola le enviamos el mensaje utilizando el método Send de la cola, que recibe un mensaje como parámetro.

Esperar la respuesta por la cola de salida:

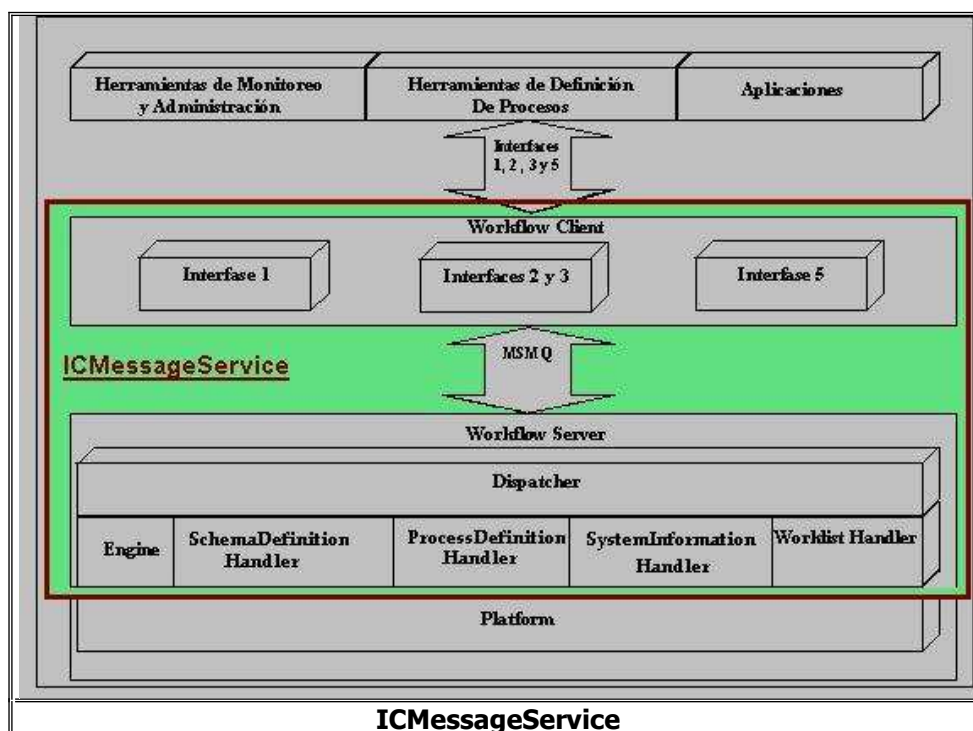
El método que envíe los mensajes a la cola de MSMQ, recibirá un parámetro que le indique si el método requiere o no esperar por una respuesta desde el servidor. En caso afirmativo, deberemos crear otro objeto del tipo MessageQueue y luego de enviar el mensaje de solicitud, esperar en dicha cola por la respuesta al mismo. Al enviar la solicitud, MSMQ le asigna un identificador al mensaje. Por tanto el proceso quedará esperando una respuesta para ese identificador de mensaje en la cola de salida del servidor.

Procesar la respuesta:

En el cuerpo del mensaje de respuesta vendrá un XML, con un único nodo, conteniendo en su texto la respuesta. Este string contenido en el mensaje de respuesta, será devuelto por el método que encapsule la comunicación y el método expuesto por el cliente sabrá obtener, en cada caso, la respuesta a partir de este string. Si, por ejemplo el método llamado es "Connect", el cliente podría obtener de la respuesta el identificador de sesión, si fuera "Disconnect", la respuesta será ignorada, si fuera un método que requiera un DataSet por respuesta, este string podría ser la serialización del mismo, etc., etc.

3.2 ICMesaService

ICMessageService es un servicio que permite administrar cualquier número de colas de mensajes de Message Queue. En particular, en nuestro caso estará administrando las colas de intercambio de mensajes entre el Cliente de WorkFlow y el Dispatcher, pero la implementación permite administrar cualquier tipo de colas de mensajería para cualquier uso.



Estructura de la aplicación

La clase principal de la aplicación es **ServiceControl**, clase que hereda de **ServiceBase**. Como se puede ver en el anexo de programación de servicios de Windows en .NET Framework, ServiceBase es la clase con la que, en las librerías del Framework, se representa el concepto de servicio. Al heredar de **ServiceBase**, se deben implementar los métodos **OnStart** y **OnStop**, además de los métodos opcionales **OnPause** y **OnContinue**.

La clase **ServiceControl**, al ser instanciada, crea una serie de objetos **WorkerInstance** (que se crean para cada cola de mensajes que requiere procesamiento). A su vez, la clase **WorkerInstance** crea una serie de objetos **WorkerThread** (tantos como el número necesario de subprocesos definido para procesar la cola en el archivo de configuración). La clase **WorkerThread** crea un subproceso de procesamiento que realiza el trabajo real del servicio.

El propósito principal de las clases **WorkerInstance** y **WorkerThread** es la confirmación de los comandos Iniciar, Detener, Pausar y Continuar del control de servicios (ServiceControl). Finalmente, ya que estos procesos no se deben bloquear, las acciones de los comandos exigirán una acción en un subproceso de procesamiento en segundo plano.

WorkerThread es una clase abstracta heredada por **WorkerThreadAppSpecific**, **WorkerThreadRoundRobin**, **WorkerThreadDisperce** y **WorkerThreadAssembly**. Cada una de estas clases procesa mensajes de forma diferente. Las dos primeras procesan mensajes enviándolos a otra cola (la diferencia está en la forma de determinar la cola receptora), la tercera manda el mensaje a toda una serie de colas de mensajería, mientras que la última utiliza las propiedades de mensaje para llamar a un método de componente. Este será el caso de estudio que nos convoque con mayor detalle, dado que es el caso utilizado particularmente en la aplicación que administra la mensajería intercambiada entre el cliente de WorkFlow y el Dispatcher (cara visible del lado del servidor). El Dispatcher es un Assembly (una aplicación

en forma de dll) a la que el `ICMessageService` le dirá que procese cada mensaje que este reciba por la cola de mensajes entrantes del `Workflow`.

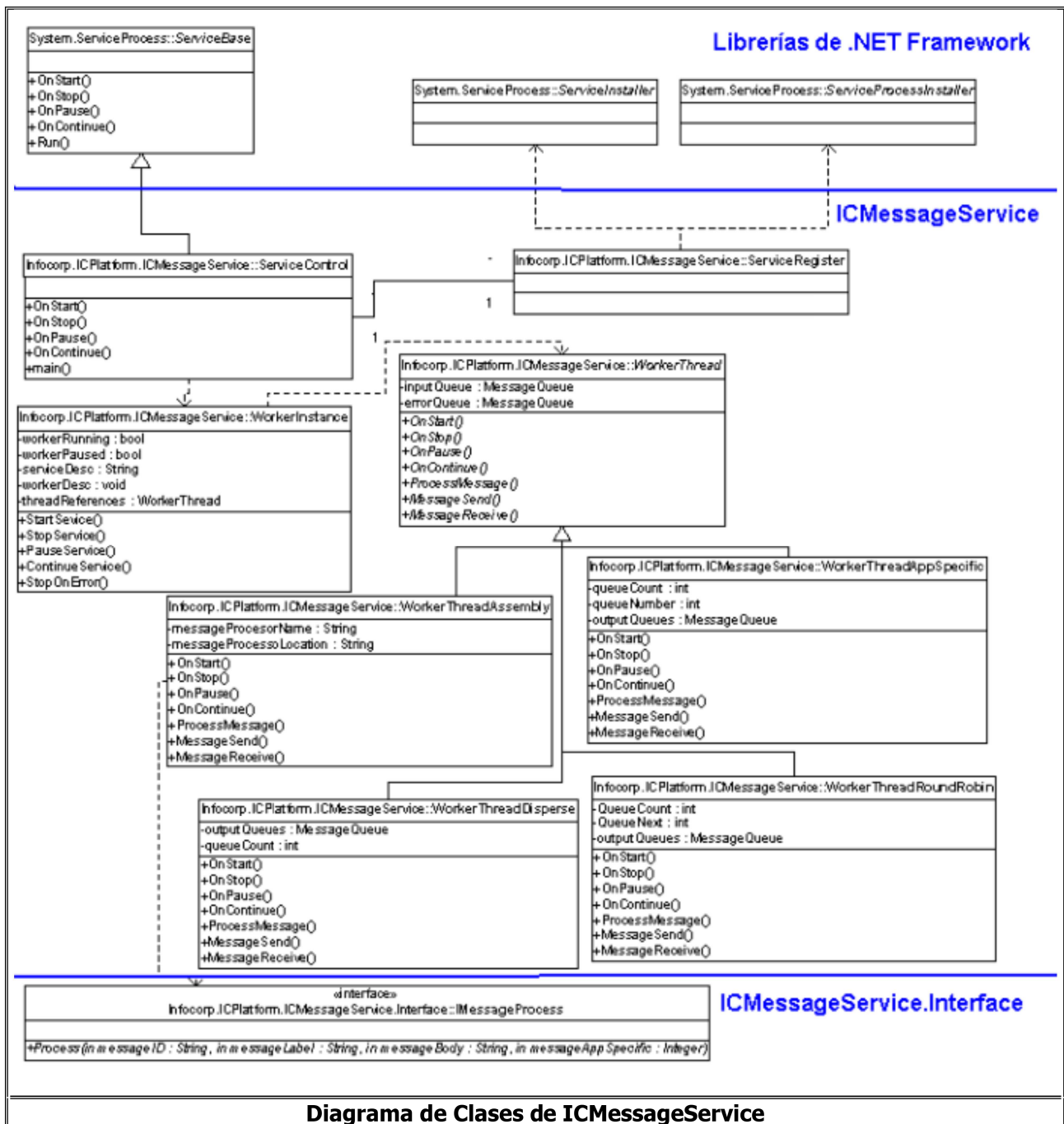
Control de Errores

El control de errores de .NET Framework se basa en una clase base **Exception**. Cuando se provocan o interceptan errores, deben ser de una clase derivada de **Exception**. La clase **WorkerThreadException** representa una implementación de este tipo, que extiende la clase base agregando una propiedad que define si el servicio debe seguir ejecutándose.

Interfaz `IMessageProcess`

Una de las implementaciones proporcionadas de **WorkerThread** es una clase que llama a un método de componente. Esta clase denominada **WorkerThreadAssembly**, utiliza una interfaz, `IMessageProcess`, para definir el contrato entre el servicio y el componente.

Se designa el método **Process**, para procesar mensajes. El tipo de enumeración **ProcessMessageReturn** define el código devuelto desde el método **Process**. Las definiciones de la enumeración es la siguiente: *Good* continúa el procesamiento, *Bad* escribe el mensaje en la cola de errores y *Abort* finaliza el procesamiento.



Configuración del servicio:

Como se mencionó antes, la estructura básica de un servicio es una clase que hereda de **ServiceBase**, la **ServiceControl**. Los métodos importantes son **OnStart**, **OnStop**, **OnPause** y **OnContinue**; cada método reemplazado se corresponde directamente con una acción de control del servicio. El propósito del método **OnStart** es crear objetos **WorkerInstance**. A su vez, la clase **WorkerInstance** crea objetos **WorkerThread** a partir de los cuáles se crean los subprocesos que realizan el trabajo del servicio. La configuración en tiempo de ejecución del servicio y, por tanto, las propiedades de los objetos **WorkerInstance** y **WorkerThread**, se mantiene en un archivo de configuración XML.

El nombre del archivo de configuración XML se incluye en un archivo de configuración de la aplicación. Los archivos de configuración de la aplicación contienen los valores de configuración que incluyen la directiva de enlace de ensamblados, los objetos remotos, los canales personalizados y demás valores que la aplicación

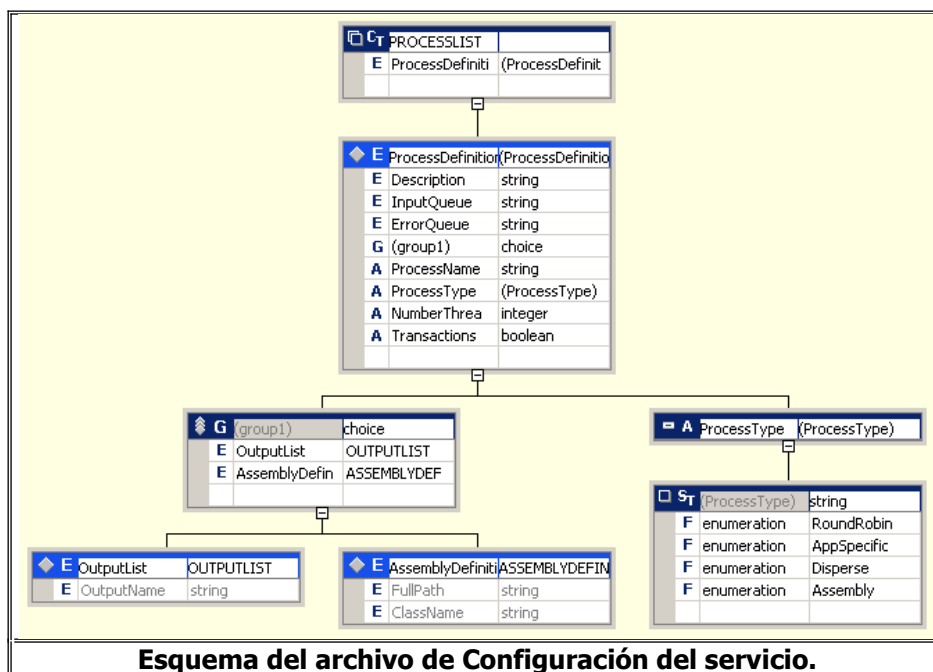
puede leer. El nombre del archivo de configuración de la aplicación se forma con el nombre de la aplicación y una extensión .config; el archivo se guarda en la carpeta ejecutable de la aplicación. Por ejemplo, la aplicación del servidor de mensajes se denomina ICMessagingService.EXE y el archivo de configuración ICMessagingService.EXE.config. Dentro del archivo de configuración de la aplicación, una sección especial denominada appSettings permite obtener una configuración que la aplicación puede leer. Esta configuración es un conjunto de pares de nombre y valor. En este caso, una clave denominada ConfigurationFile proporciona la ruta de acceso completa al archivo de configuración XML. De éste se obtiene la parametrización del servicio.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <AppSettings>
    <add key="ConfigurationFile" value="C:\Infocorp\ICPlatform\
ICMessagingService\MessagingService.xml" />
    <add key="SchemaFile" value="C:\Infocorp\ICPlatform\
ICMessagingService\MessagingService.xsd" />
  </AppSettings>
</configuration>
```

El anterior es el archivo de configuración de la aplicación ICMessagingService.exe. De esta forma la configuración del servicio es totalmente parametrizable, sin necesidad de retocar el código. Por ejemplo, vemos que el archivo de configuración establece que el archivo de configuración de los servicios es el MessagingService.xml. En caso que queramos modificar la configuración del mismo generamos un nuevo XML de configuración y cambiamos aquí su nombre. A continuación estudiaremos un poco más al detalle el archivo de configuración de la aplicación y el esquema que este sigue.

Esquema del archivo de configuración:

El esquema del archivo de configuración es simple y a continuación observamos una imagen generada con el administrador de esquemas XSD de .NET Framework, sobre la cual, basamos nuestro análisis. Observemos primero el archivo.



El esquema básicamente consiste en una lista de Definiciones de procesos. Cada definición de proceso tiene una serie de atributos y elementos. Los atributos son:
 ProcessName: Nombre del proceso que atiende la cola.

ProcessType: Tipo del proceso. Enumerado con los cuatro valores vistos anteriormente (RoundRobin, AppSpecific, Disperse, Assembly).

NumberThreads: Números de hilos de ejecución que se desea crear, cada uno con una instancia del proceso, atendiendo todos la cola de MSMQ respectiva.

Transaction: Indicador de si el procesamiento de los mensajes se hace o no de forma transaccional.

Aparte de estos atributos, según sea el tipo de proceso, tiene más elementos. Si el tipo de proceso es Assembly, tiene un elemento del tipo AssemblyDefinition y, en caso contrario (cualquiera de los otros tres tipos de proceso) tiene un elemento del tipo OutputList. Este último tipo tiene multiplicidad aceptada entre 1 y N, y cada instancia tiene un único atributo de tipo string, que establece el nombre de la cola de salida. En caso de ser un elemento del tipo AssemblyDefinition, tiene dos strings, el path a Assembly y el nombre de la clase, dentro del assembly, que implementa la interfaz IMessageProcess.

Archivo de configuración para la administración de la comunicación Cliente – Servidor:

```
<?xml version="1.0" encoding="utf-8" ?>
<ProcessList xmlns="urn:messageservice-schema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:messageservice-schema MessageService.xsd">
  <ProcessDefinition ProcessName="ICWorkflowProcess" ProcessType="Assembly"
    NumberThreads="5" Transactions="false">
    <Description>ICWorkflow - Assembly Worker</Description>
    <InputQueue>.\private$\icworkflow_in</InputQueue>
    <ErrorQueue>.\private$\icworkflow_error</ErrorQueue>
    <AssemblyDefinition>
      <FullPath>C:\Infocorp\ICSolutions\ICWorkflow\ICWorkflowDispatcher\ICWorkflowDispatcher.dll</FullPath>
      <ClassName>Infocorp.ICSolutions.ICWorkflow.ICWorkflowDispatcher</ClassName>
    </AssemblyDefinition>
  </ProcessDefinition>
</ProcessList>
```

MessageService.XML

En nuestro caso el único proceso que tendremos leyendo de las colas de MSMQ será el proceso del procesamiento de los mensajes al motor de Workflow. Por tanto, en los atributos del único proceso definido en la lista, establecemos que es no transaccional, de tipo assembly y con cinco hilos de ejecución paralelos. Además, por ser de tipo assembly, debemos establecer el path de acceso a la dll y, dentro de ésta el nombre de la clase que implementa la interfaz IMessageProcess.

En base a este archivo, la clase service control sabe que tiene un único proceso a administrar, llamado ICWorkflowProcess. Por tanto agregará a su colección de procesos a administrar solamente éste. Para el mismo, sabrá también que se desea llevar a cabo la atención de la cola de entrada con cinco hilos de ejecución. Como el XML de configuración establece que el tipo de proceso es un assembly, el ServiceControl sabrá que debe instanciar cinco WorkerThread del tipo WorkerThreadAssembly. Para instanciar ese assembly, el archivo también indica donde está la dll y como se llama la clase. En nuestro caso, hemos declarado un proyecto del tipo dll, donde tenemos el Dispatcher, que se llama ICWorkflowDispatcher. Dentro de este proyecto, está definida la clase Infocorp.ICSolutions.ICWorkflow.ICWorkflowDispatcher, que es la implementación del Dispatcher propiamente dicha. El Dispatcher implementa la interfaz IMessageProcess, por lo que define el método process al que cada objeto del tipo WorkerThreadAssembly, le pasa los datos del mensaje a procesar.

Las clases WorkerInstance y WorkerThread

Tienen los métodos de control de servicio correspondientes que se llamarán en función de la acción de control del servicio. Como se hace referencia a cada objeto **WorkerInstance** en la colección Hashtable, el contenido de esta colección está enumerado a fin de llamar al método de control de servicio adecuado:

```
foreach (WorkerInstance WorkerInstance in mWorkerInstances.Values)
  WorkerInstance.Start();
```

De forma similar, los métodos **OnPause**, **OnContinue** y **OnStop** implementados llaman a los métodos correspondientes en los objetos **WorkerInstance**.

La clase WorkerInstance

La función principal de la clase **WorkerInstance** es crear y administrar objetos **WorkerThread**. Los métodos **Start**, **Stop**, **Pause** y **Continue** llaman a los métodos de **WorkerThread** correspondientes. Los

verdaderos objetos **WorkerThread** se crean en el método **Start**. Como la clase **Service**, que utiliza una colección **Hashtable** para administrar las referencias a los objetos de trabajo, **WorkerInstance** utiliza un objeto **ArrayList**, un vector cuyo tamaño se ajusta dinámicamente para mantener una lista de objetos de subproceso.

La clase **WorkerInstance** creará en este vector, una de las versiones implementadas de la clase **WorkerThread**. La clase **WorkerThread**, que se describe a continuación, es una clase abstracta que debe heredarse. La clase derivada define la forma de procesar los mensajes.

Cuando se hayan creado todos los objetos, podrán iniciarse llamando al método **Start** de cada objeto de subproceso:

```
foreach(WorkerThread thread in mThreads)
    thread.Start();
```

Los métodos **Stop**, **Pause** y **Continue** realizan operaciones similares en un bucle **foreach**.

La clase abstracta **WorkerThread**

WorkerThread es una clase abstracta que se hereda por **WorkerThreadAppSpecific**, **WorkerThreadRoundRobin**, **WorkerThreadDisperce** y **WorkerThreadAssembly**. Como la mayor parte del procesamiento de una cola es idéntico, independientemente de cómo se procese el mensaje, la clase **WorkerThread** proporciona esta funcionalidad. La clase proporciona métodos abstractos que deben reemplazarse para administrar recursos y procesar mensajes.

El trabajo de la clase, se implementa de nuevo en los métodos **Start**, **Stop**, **Pause** y **Continue**. El método **Start** hace referencia a las colas de entrada y de errores.

Cuando se han definido las referencias de cola de mensajes, se crea un subproceso denominado **ProcessMessages** para la función de procesamiento real. En .NET Framework, el subprocesamiento se realiza fácilmente mediante el espacio de nombres **System.Threading**.

La función **ProcessMessages** es un bucle de procesamiento basado en un valor de tipo **Boolean**. Cuando se establece en **false**, finaliza el bucle. Por tanto, el método **Stop** del objeto de subproceso sólo establece este valor de tipo **Boolean** y después une el subproceso al subproceso principal, además de cerrar las colas de mensajes abiertas

El bucle **ProcessMessages** tiene la siguiente estructura básica:

- Se recibe un objeto **Message**.
- Si se ejecuta correctamente el método **Receive** de un objeto **Message**, el objeto llama al método abstracto **ProcessMessage**.
- Si no se pueden ejecutar correctamente los métodos **Receive** o **ProcessMessage**, se envía el objeto **Message** a una cola de errores.

Las clases derivadas de **WorkerThread**

Cualquier clase que herede de **WorkerThread** debe proporcionar métodos **OnStart**, **OnStop**, **OnPause**, **OnContinue** y **ProcessMessage**. Los métodos **OnStart** y **OnStop** tienen como objetivo adquirir y liberar recursos de procesamiento. Los métodos **OnPause** y **OnContinue** se incluyen para permitir la liberación temporal y readquisición de estos recursos. El método **ProcessMessage** debe procesar un mensaje y provocar una excepción **WorkerThreadException** en caso de error.

Como el constructor **WorkerThread** define parámetros en tiempo de ejecución, las clases derivadas deben llamar al constructor de la clase base.

Se proporcionan clases derivadas para dos tipos de procesamiento:

- enviar mensajes a otra cola
- llamar a un método de componente.

Las dos implementaciones que reciben y envían mensajes utilizan una técnica de turnos o de alternancia de aplicaciones, contenida en la propiedad **AppSpecific** de mensaje, como factor determinante de la cola que

se va a usar. En este escenario el archivo de configuración debe contener una lista de rutas de cola. Los métodos **OnStart** y **OnStop** implementados deben abrir y cerrar una referencia a estas colas. La última implementación, que llama a un componente con los parámetros de mensaje, es un poco más interesante. El método **ProcessMessage** llama a un componente .NET mediante la interfaz **IMessageProcess**. Los métodos **OnStart** y **OnStop** obtienen y liberan una referencia a este componente. En este escenario el archivo de configuración deben contener dos elementos: el nombre de clase completo y la ubicación del archivo en que reside la clase. Se llama al método **Process** en el componente, tal y como se define en la interfaz **IMessageProcess**.

En la aplicación particular de este **MessageService** utilizada en nuestro proyecto, nosotros administraremos un cola de mensajes de entrada al servidor de **WorkFlow**, que distribuirá los mensajes entre las 5 (en principio) instancias de la clase **WorkerThreadAssembly**. Cada una de éstas, tendrá una referencia a la clase que implementa la interfaz, que en nuestro caso nuevamente, es el **ICWorkFlowDispatcher**. Por tanto, el resultado que obtendremos nosotros con la configuración que hemos dado a nuestro servicio **ICMessageService**, consiste en cinco instancias en paralelo del **Dispatcher**, atendiendo las solicitudes de los posibles clientes a través de una cola de **MSMQ**. El análisis detallado del **Dispatcher** se hará más adelante en este mismo documento, pero en principio podemos establecer que el **Dispatcher** tiene una referencia al **WorkListHandler** y otra a una instancia del **WFServer**, por lo que tendremos también cinco instancias de cada uno de estos trabajando en paralelo.

Instrumentación

La plataforma .NET Framework ha simplificado en gran medida la inclusión del registro de eventos, los contadores de rendimiento y **Windows Management Instrumentation (WMI)** en las aplicaciones. La aplicación de mensajería utiliza el registro de eventos y los contadores de rendimiento del ensamblado **System.Diagnostics**.

En la clase **ServiceBase**, se puede habilitar el registro automático de eventos. Además, el miembro **EventLog** de **ServiceBase** permite escribir en el registro de eventos de la aplicación.

Para que una aplicación pueda escribir registros de eventos distintos del de la aplicación, puede crear y obtener fácilmente una referencia a un origen **EventLog**, de la misma manera que en la clase **WorkerInstance** y puede utilizar el método **WriteEntry** para registrar entradas de registro.

En .NET Framework se simplifican en gran medida los contadores de rendimiento. Esta aplicación de mensajería proporciona contadores que hacen un seguimiento del número de mensajes procesados por segundo para cada subproceso de procesamiento, el proceso de trabajo del que se derivan los subprocesos y la aplicación como un todo. Para proporcionar esta funcionalidad, se deben definir las categorías de contador de rendimiento y, a continuación, incrementar las correspondientes instancias de contador.

Las categorías de contadores de rendimiento del método **OnStart** se definen dentro del servicio. Estas categorías representan los dos contadores: el número total de mensajes y el número de mensajes procesados por segundo:

Instalación

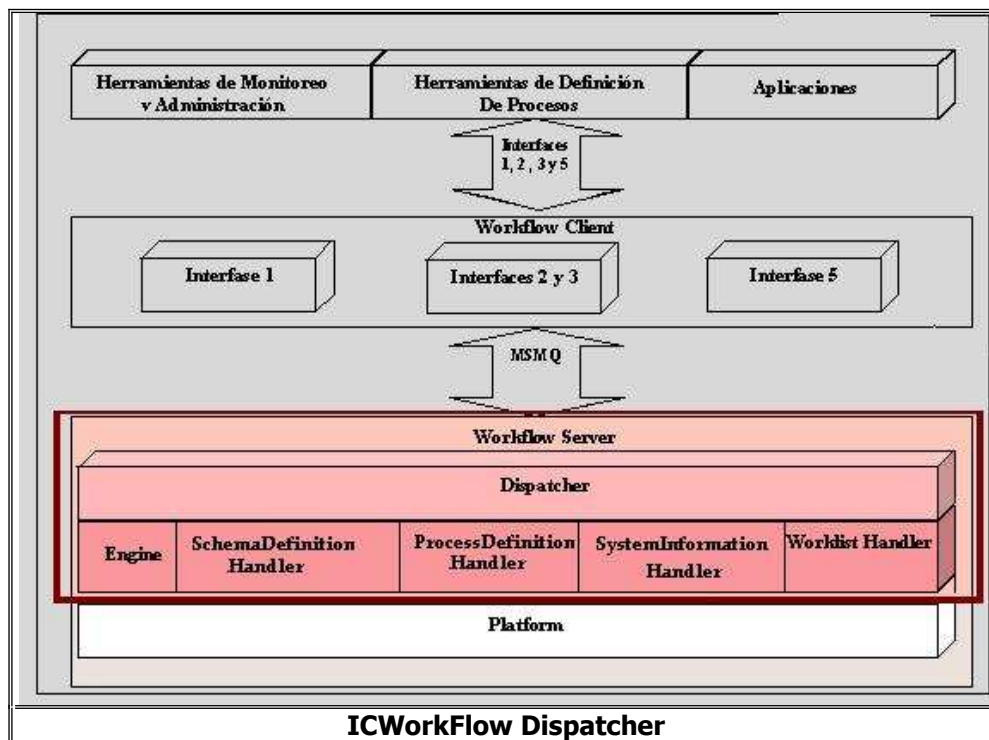
Como todo servicio de Windows, este debe ser instalado, antes de lanzarse a funcionar. Como esta aplicación es un servicio de Windows, debe instalarse mediante **installutil.exe**. Para facilitar esta instalación, es necesario utilizar una clase que herede de la clase **Installer** del ensamblado **System.Configuration.Install**:

Para un servicio de Windows hay que utilizar un instalador para el servicio y otro instalador para el proceso del servicio, a fin de definir la cuenta con la que se ejecutará el servicio. Otros instaladores permiten el registro de recursos como registros de eventos y contadores de rendimiento.

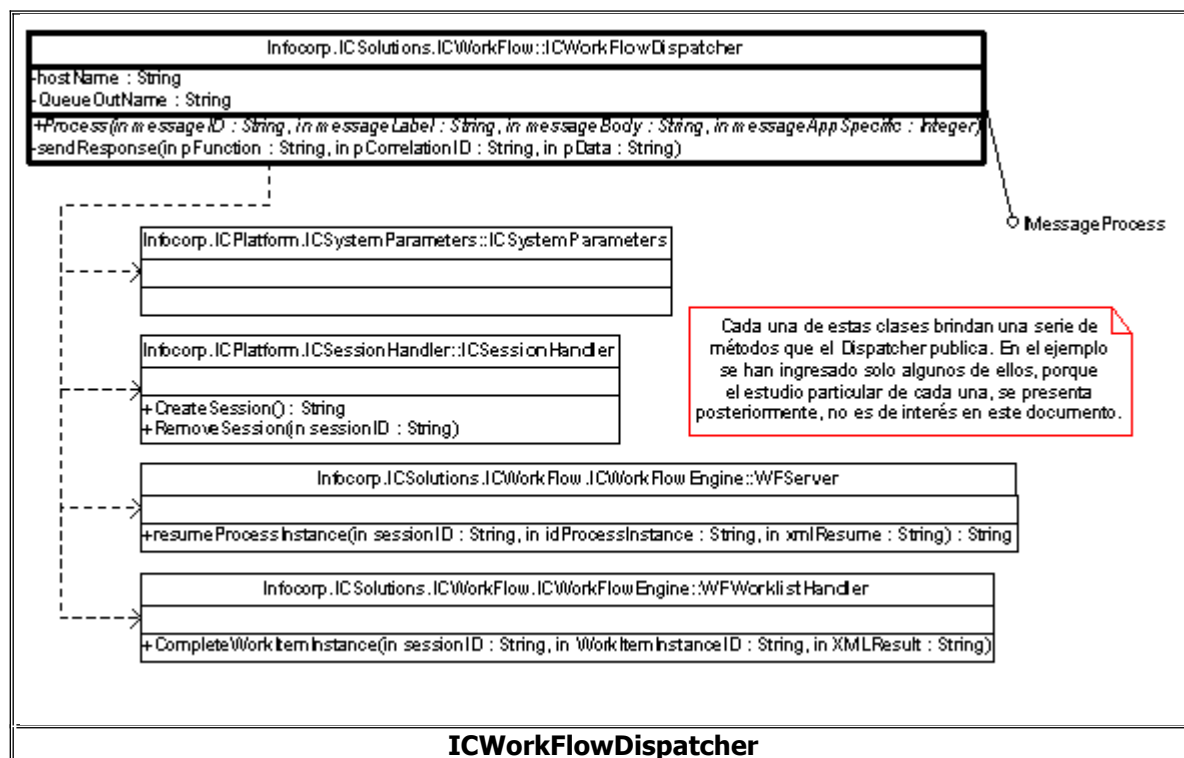
3.3 ICWorkflow Dispatcher

El WorkflowDispatcher es la contraparte del lado del servidor del ICWorkflowClient.

Como vimos en la descripción del cliente, por cada método que éste expone, genera un mensaje de MSMQ, donde el asunto del mensaje identifica la funcionalidad en cuestión y en el cuerpo del mismo es un XML, con los parámetros de la llamada cargados en un hash y serializados como elemento del documento.



¿Qué debe entonces hacer el Dispatcher? Debe, a partir de los datos de un mensaje, identificar la funcionalidad, obtener la serialización del hash con los parámetros, deserializar el Hash y luego, según la funcionalidad buscar los valores de los parámetros en el mismo. De acuerdo a la funcionalidad, determina también a qué componente del motor derivar la llamada. Con toda esta información, deriva la llamada al que corresponda, a la funcionalidad debida y con los parámetros adecuados.



El Dispatcher resuelve las solicitudes que recibe, principalmente en función del Engine y el WorkListHandler. Administra también la conexión y desconexión de los clientes, para lo cual utiliza al SessionHandler. Por tanto, tendrá como miembros de clase,

- una referencia al WFEEngine (motor de WorkFlow),
- una referencia al WorkListHandler (manejador de la WorkList)
- una referencia al ProcessDefinitionHandler (manejador de las definiciones de procesos)
- una referencia al SystemInformationHandler (manejador de los parámetros y páginas del sistema)
- una referencia al SchemaDefinitionHandler (manejador de la definición de esquemas)
- una referencia al SessionHandler (manejador de sesiones que abre y cierra el cliente).

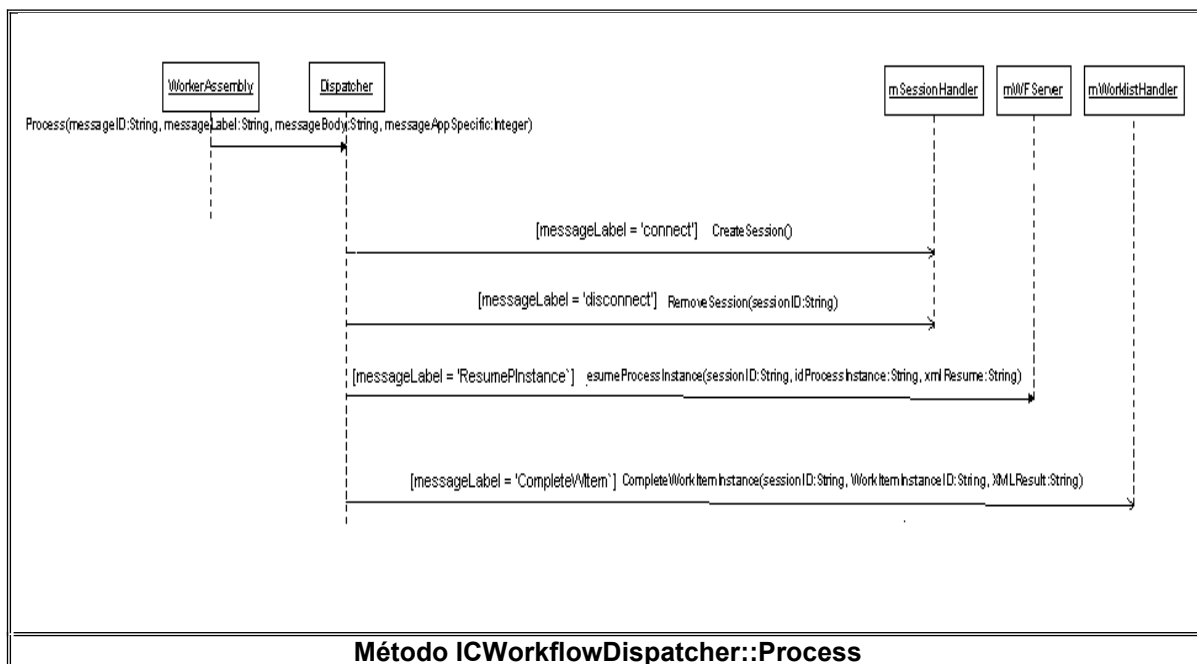
Cada uno de estos manejadores, se parametriza mediante los parámetros del sistema que se administran por medio de los ICSystemParameters. Cada Dispatcher que se crea, por tanto se hace en referencia a un conjunto de parámetros. La idea es cargar los parámetros del sistema, a partir del identificador único del sistema en cuestión. Los parámetros del sistema se "saben" cargar a partir del identificador único. Cuando se cargan, levantan la parametrización del SessionHandler, del WorkListHandler y del WFServer. Por tanto, una vez cargados los parámetros del sistema, el Dispatcher crea cada uno de los anteriores pasándoles su parametrización en el constructor.

Método Process

La funcionalidad básica del Dispatcher es la descrita anteriormente. Debe implementar la interfaz IMessageProcess para poder consumir los mensajes que recibe del cliente. Todo este trabajo es el que el Dispatcher realiza ante cada mensaje que procesa. El método llamado Process, recibirá el identificador del mensaje, el label (o lo que sería el subject de un mail) y el cuerpo del mensaje.

A partir de estos datos, toma una serie de decisiones. Como primer paso, según el label del mensaje (descriptor de la funcionalidad) decide que método se está invocando y a quién debe derivarse. Decide entre invocar al SessionHandler, al WFServer o al WorkListHandler. Luego debe obtener del cuerpo del mensaje el elemento que tiene la "data", es decir, los parámetros de la llamada a la funcionalidad que estemos procesando. Deserializando este texto, obtendrá el hash con los parámetros del método. Del hash y conociendo el método, obtiene los parámetros y hace la llamada respectiva.

Con esos parámetros llama al método adecuado, del miembro adecuado y le pasa los parámetros respectivos. A continuación vemos un diagrama de colaboración, mostrando una llamada al Dispatcher.



Respuestas del Dispatcher al Cliente

Hay dos tipos de respuesta del Dispatcher hacia sus invocadores. Por un lado, el Dispatcher será directamente invocado por el servicio de Windows que controla el funcionamiento de las colas de Message Queue (ICMessageService). Recordemos que uno de los tipos de consumidores de mensajes que soporta el ICMMessageService es el tipo Assembly. Al crear un controlador de mensajería de tipo assembly el WorkerThread respectivo, ante cada mensaje recibido llamará al método Process de una interfaz que debe cumplir todo el que se desee suscribir como assembly consumidor de los mensajes que lleguen a estas colas. El Dispatcher implementa la interfaz IMessageProcess, por lo que ante cada invocación al método Process, devuelve un valor correspondiente al enumerado ProcessMessageReturn:

- ReturnGood
- ReturnBad
- ReturnAbort

Por otro lado el Dispatcher, derivada la llamada al objeto correspondiente (al servidor, al WorkList Handler o al Session Handler), en los casos en que éstos devuelven un resultado, el Dispatcher lo serializa y lo pone como valor de un tag XML, que se guarda en el body de un mensaje de MSMQ. Ese mensaje es enviado a la cola de salida (OUT).

En los casos en que el cliente está esperando la respuesta, la capturará y actuará en consecuencia. En caso de provocarse un error, en el XML del body va el mensaje original y la descripción del error capturado.

4 MOTOR DEL SISTEMA DE WORKFLOW.

El motor del sistema de Workflow, es el encargado de gestionar la ejecución de las instancias de procesos. Con tal motivo, su principal funcionalidad es la de interpretar la definición de proceso, realizado con el editor de flujos, e ir ejecutando el mismo.

A continuación se enumerarán las funcionalidades brindadas por el motor de Workflow.

- **Start Process:** Se encarga de hincar una nueva instancia de un proceso. Recibe entre otros parámetros la identificación del proceso y el esquema de los datos de entrada a dicho proceso.
- **Resume Process:** Es el encargado de continuar la instancia de un proceso que había sido pausado.
- **Restart Process:** Reinicia la instancia de un proceso, deteniendo la instancia actual e iniciando una nueva instancia.
- **Stop Process:** Detiene la ejecución de un proceso.
- **Suspend Process:** Suspende o pausa la ejecución de un proceso.

Pero el motor nos solo brinda funcionalidades sobre los procesos, sino que también debe brindar funcionalidades sobre las instancias de las actividades, ya que un usuario puede, o bien tomar una acción sobre un proceso, o sobre una actividad determinada. Por este motivo también brinda las siguientes funcionalidades.

- **Start Activity:** Se encarga de hincar una nueva instancia de un proceso. Recibe entre otros parámetros la identificación del proceso y el esquema de los datos de entrada a dicho proceso.
- **Resume Activity:** Es el encargado de continuar la instancia de un proceso que había sido pausado.
- **Stop Activity:** Detiene la ejecución de un proceso.
- **Suspend Activity:** Suspende o pausa la ejecución de un proceso.

Para poder realizar los métodos descriptos, el motor del sistema de Workflow, utiliza información, mediante el uso de los siguientes objetos:

Process Data:

Esta clase cuenta con dos atributos de tipo HashTable:

- **GlobalData:**
 - Variables globales del proceso (Datafields)
- **StateData:**
 - Definición del esquema con el cual fue creado el proceso
 - Identificador de la instancia de proceso
 - Estado del proceso
 - Fecha de apertura
 - Resultado de ejecución de la última actividad
 - Estado de la ejecución de la última actividad
 - XML de entrada al proceso
 - Identificador de la actividad actual
- **UserData:**

**Relevant
Data**

- Xml que ingresa el usuario en una actividad de tipo participant

Process Instante:

Este objeto representa la instancia de un proceso, con tal motivo, contiene la siguiente información:

- **Process Definition:** La definición del proceso que se está ejecutando, dicha definición está conformada, por los objetos descritos en el esquema (XSD) del lenguaje de definición de procesos (XPDL)
- **Process Data:** Contiene la información anteriormente descrita.

Debido a las diferentes funcionalidades brindadas por el motor sobre una instancia de un proceso, el estado en el que se encuentra, varía según su ejecución o según las acciones tomadas sobre el mismo. Por tal motivo, se definen un conjunto de estados que se enumeran a continuación:

- **Open.Running:** Indica que el proceso está ejecutándose sin inconvenientes.
- **Open.NotRunning.NotStarted:** Indica que el proceso no comenzó a ejecutarse aún, ya que no fue iniciado, pero está en proceso.
- **Open.NotRunning.Suspended:** El proceso fue suspendido por un usuario.
- **Open.NotRunning.Suspended.byActivity:** El proceso se ha suspendido, por que la última actividad a ejecutar fue suspendida.
- **Open.NotRunning.Suspended.bySystem:** El proceso se ha suspendido externamente, por un usuario del sistema.
- **Closed.Terminated:** El proceso fue terminado manualmente por un usuario del sistema.
- **Closed.Aborted:** El proceso fue abortado por errores en la ejecución de actividades o toma de decisiones.
- **Closed.Completed:** El proceso terminó de ejecutarse correctamente.
- **Closed.Restarted:** El proceso fue terminado para ser reiniciado.

A su vez, las actividades que se van ejecutando, también poseen un estado, en particular la última actividad ejecutada, lo que permite saber si el proceso puede seguir correctamente o tuvo algún problema. A continuación se señalan los diferentes estados en los que se puede encontrar una actividad.

- **Open.Running:** La actividad está siendo ejecutada en este momento.
- **Open.Suspended.byStartMode:** Significa que la actividad fue suspendida por que su comienzo debe ser realizado manualmente.
- **Open.Suspended.byExecution:** Significa que la actividad a ejecutar es asincrónica.
- **Open.Suspended.byFinishMode:** La actividad es suspendida por tener un final manual
- **Close.Aborted:** La actividad fue abortada por un error en su ejecución
- **Close.Terminated:** Indica que la actividad fue terminada por un usuario
- **Close.Completed:** La actividad fue terminada con éxito.

Una vez señaladas las principales características del motor de workflow, se está en condición de señalar el funcionamiento de los métodos principales que el motor brinda, con la finalidad de clarificar las ideas.

Start Process:

Para iniciar un nuevo proceso de negocio el motor se crea una instancia del mismo según su definición, seteando su estado como "Open.NotRunning.NotStarted" y persistiendo dicha instancia en la base de datos. Busca la primera actividad del proceso, seteandola como actividad actual del mismo, y se ejecuta el proceso modificando su estado a "Open.Running".

Al ejecutar el proceso, el motor toma la actividad actual y verifica que no haya sido pausada o finalizada, en este caso, se ejecuta la actividad actual. Antes de su ejecución, debe traducir todos los parámetros que ésta reciba, y que estén dados en el lenguaje que permite realizar expresiones en función del globalData o de datos de ingresados al proceso, sustituyéndolos por su valor en la instancia del proceso (Por más información sobre dicho lenguaje y el mecanismo de traducción ver Apéndice S - Lenguaje para referenciar información del proceso). Luego de ejecutar dicha actividad, verifica el estado en que termino la misma. En caso de que el estado haya sido: "Closed.Aborted" o "Closed.Terminated", el proceso es finalizado anormalmente. El resultado de dicha actividad puede ser también suspendida ("Open.NotRunning.Suspended.byActivity"), en este caso suspende la ejecución del proceso, ya que la actividad es asíncrona y se debe esperar a que ésta termine.

Una vez ejecutada la actividad, se persiste nuevamente el proceso, de forma de almacenar los cambios realizados por su ejecución y se busca la nueva actividad a ejecutar. Para esto, se recorre la colección de transiciones salientes y se selecciona la primera que su condición evalúe en verdadero. Se toma la actividad final de esa transición como la siguiente a ejecutar repitiendo el procedimiento anteriormente mencionado. En caso que no haya transiciones salientes o ninguna evalúe en verdadero, el proceso aborta. La condición a evaluar, puede ser una expresión en función de variables globales o datos ingresados al proceso, según el lenguaje previamente mencionado, por tal motivo, debe traducirse la expresión antes de ser evaluada con los valores de la instancia actual del proceso. Por más información sobre el mecanismo para generar y evaluar expresiones ver apéndice sobre componentes re-usables.

Se describió anteriormente el proceso de ejecución de un flujo, se detallará ahora como se efectúa la ejecución de una actividad. Al ejecutar una nueva actividad, se crea una instancia de la misma, pasándole toda la información del proceso, seteando su estado como "Open.Running" y persistiéndola.

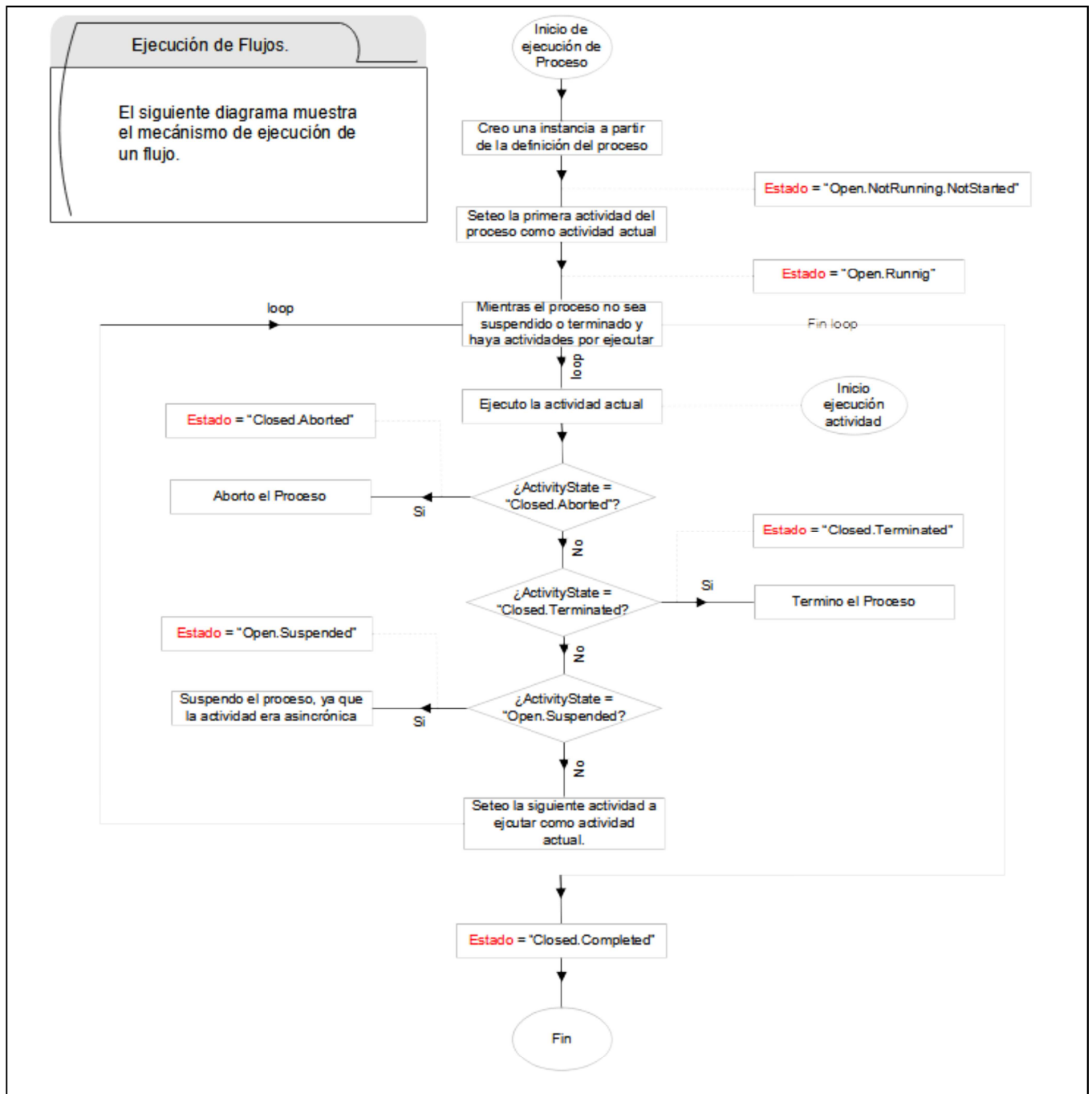
Si la actividad a ejecutar tiene un modo de comienzo manual, se suspende su ejecución, de lo contrario se ejecuta dependiendo del tipo de actividad:

- **Implementation**

- None: La única actividad de este tipo es el Milestone o hito, ya que es la única que interpreta el motor por si mismo, y lo que debe hacer es modificar el hito de la instancia del proceso.
- Tool: Este tipo de actividad invoca una aplicación externa (toolAgent). Para que dicha aplicación pueda acceder a la información de proceso debe cargarse el ActivityContext con la información anteriormente detallada, luego se invoca la aplicación (ver Apéndice S de Tool Agents). Si la aplicación se ejecuta de forma asíncrona, el proceso debe suspenderse (por ejemplo, en el caso de una tarea que debe ser realizada por un usuario), en este caso el estado de la instancia de la actividad es "Open.Suspended.byExecution". Una vez terminada la ejecución, debe actualizarse el ProcessData, según los resultados brindados por la aplicación.
- Route: Este actividad es "tonta", su finalidad es poder agregar condiciones en los flujos, de forma de representar gráficamente el camino a seguir.

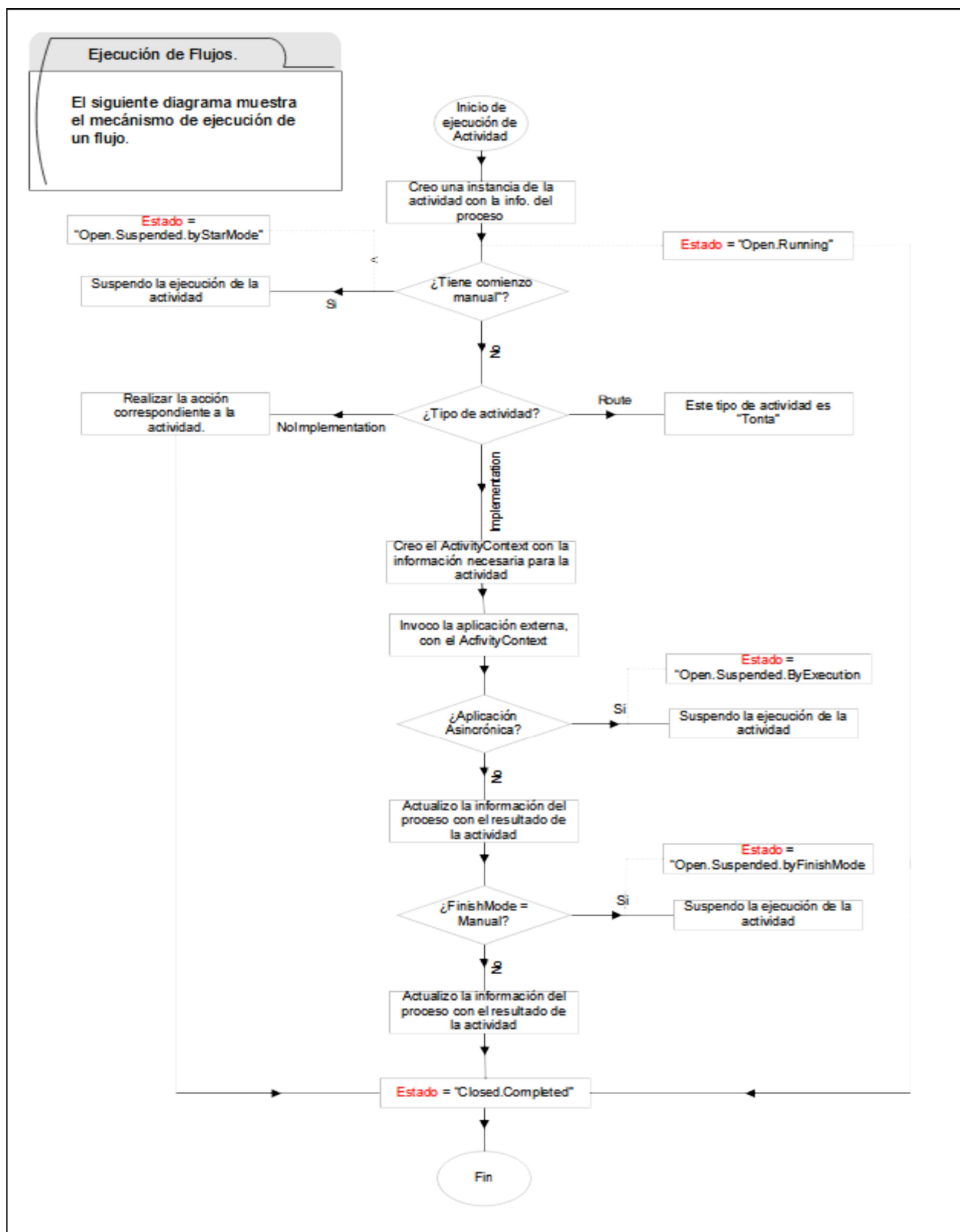
Una vez ejecutada la actividad, si su estado no fue suspendido por ejecución o por comienzo manual y la misma no debe ser finalizada manualmente (Open.Suspended.byFinishMode), se culmina con la ejecución de la misma, seteando su estado a "Closed.Completed" y persistiéndola.

En el siguiente diagrama se puede apreciar más claramente el mecanismo de ejecución de un proceso.



Se puede apreciar en el diagrama que el proceso puede terminar o bien a causa de una actividad que se suspendió, aborto o fue detenida, o bien, por que el propio proceso fue suspendido o terminado (de forma manual por un usuario). En este caso, se utiliza una variable del proceso denominada `execution_control`, la cual se setea según el proceso haya sido pausado o detenido. Por esto, antes de ejecutar la siguiente actividad se pregunta por el valor de dicha variable y se toma la acción correspondiente, seteando la variable `execution_control` en null, luego de ejecutar la acción.

El diagrama que se presenta a continuación forma parte de la ejecución de un proceso y representa la ejecución de una actividad en particular, concluyendo de esta forma el flujo de ejecución de un proceso.

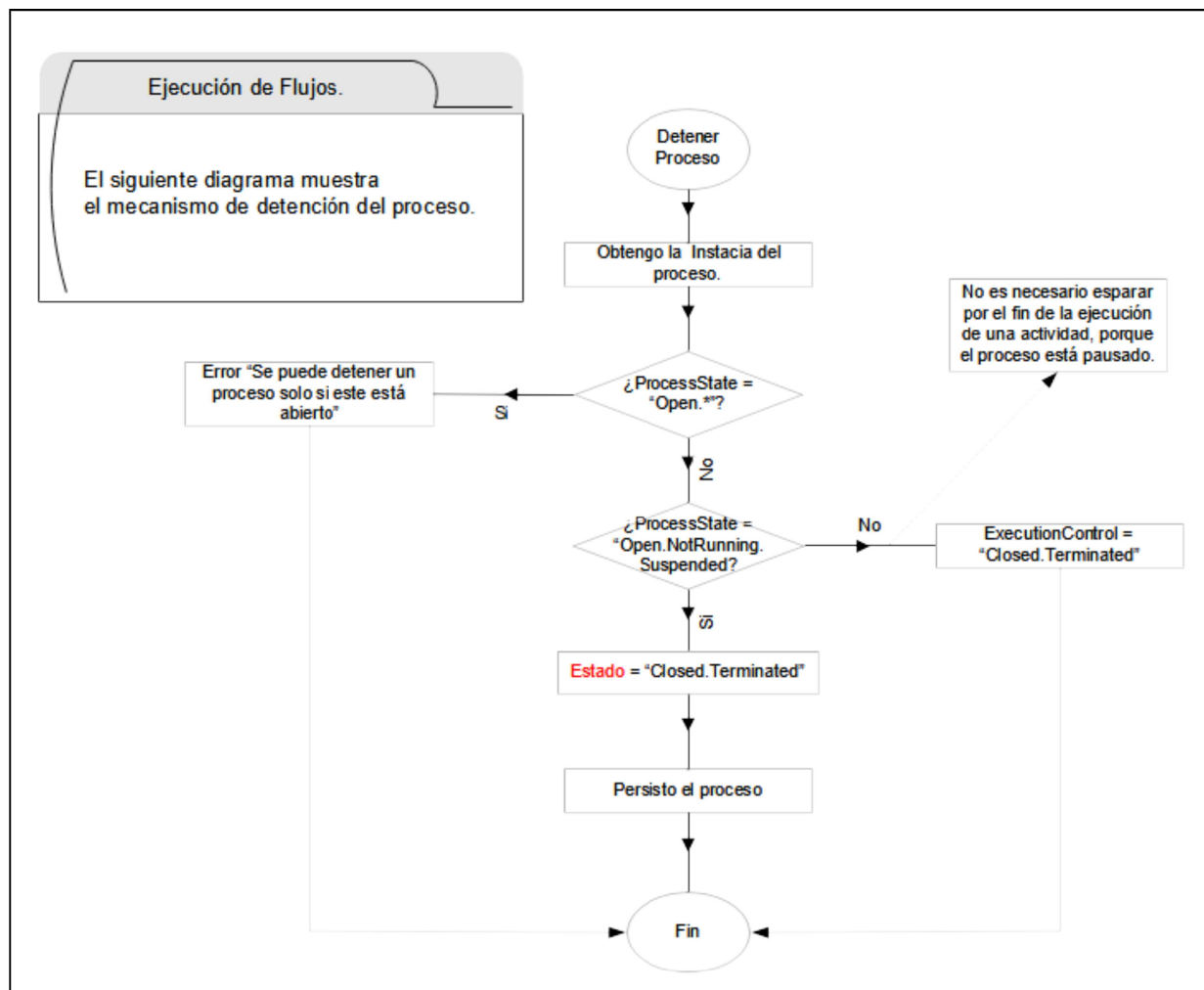


Stop Process:

Este método es el encargado de detener la ejecución del proceso por completo, impidiendo la posibilidad de continuarlo, si se desea, el mismo debería ser pausado, pero no detenido. En primera lugar, se obtiene la instancia del proceso según su identificador. Si el estado del proceso es open se procede a detener el mismo. Para cumplir tal objetivo, se setea la variable execution_control del proceso en Closed.Terminated, esta condición es chequeada durante la ejecución del proceso, de encontrarse con el estado previamente mencionado, la ejecución del mismo se detiene, modificando el estado a Closed.Terminated, y persistiendo el proceso.

Sin embargo, si el estado del proceso es open.NotRunning.Suspended, el motor no está ejecutando ninguna actividad, por lo cual no es necesario esperar que termine para suspenderlo y se puede realizar inmediatamente. Por esto, se modifica el estado del proceso a Closed.Terminated y se persiste el mismo.

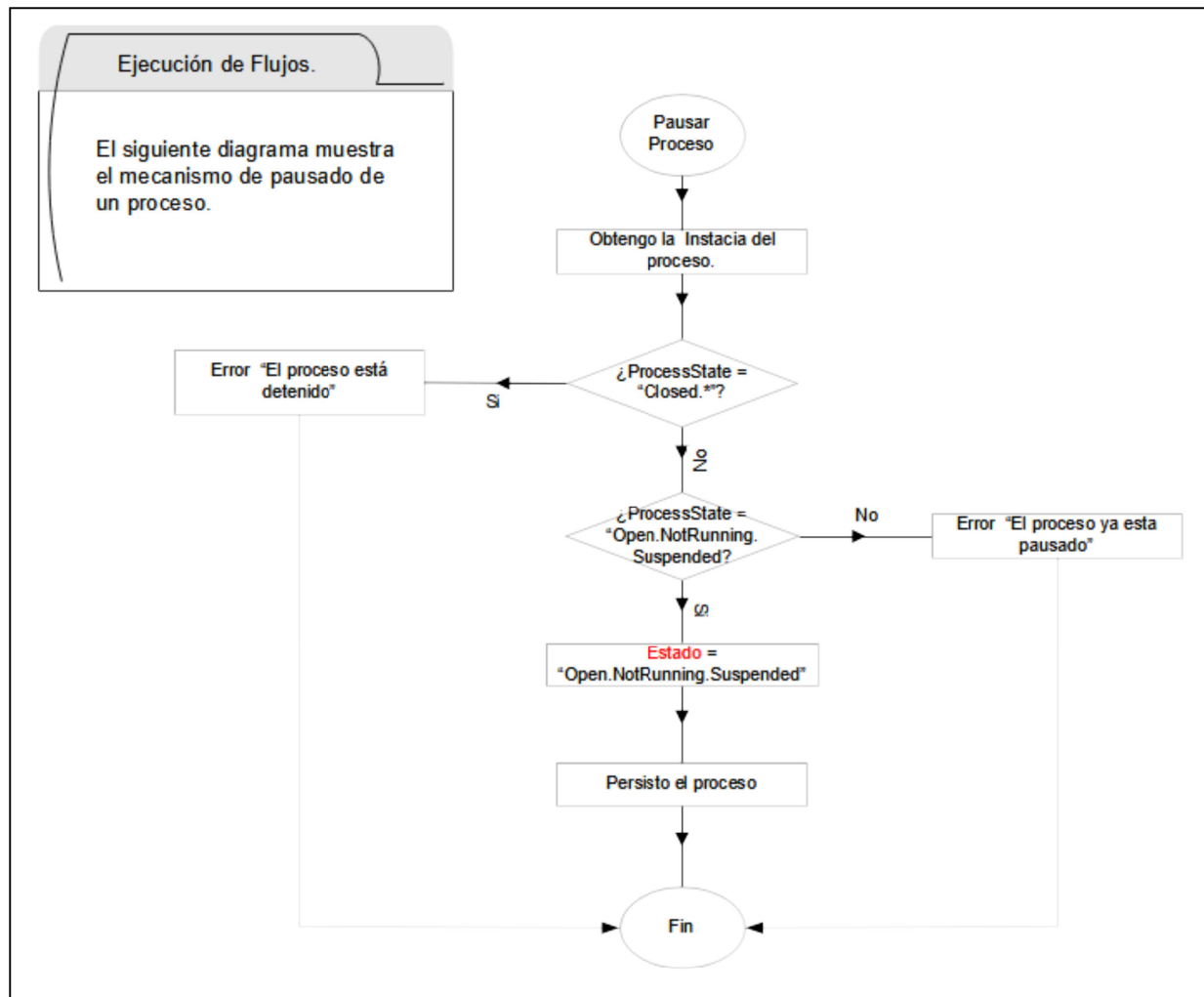
El siguiente diagrama muestra el mecanismo de detención de un proceso.



Suspend Process.

Este método es el encargado de suspender la ejecución de un proceso. Para satisfacer dicha funcionalidad, el mecanismo es similar al de detener un proceso. En primer lugar se obtiene la instancia del mismo. Si el proceso se encuentra detenido o suspendido, se envía un mensaje de error, en caso contrario, se setea la variable `execution_control` en `Open.NotRunning.Suspended`, de forma que antes de ejecutarse la siguiente actividad del proceso, se detecte el nuevo estado, suspendiendo el proceso y modificando el estado del mismo a `Open.NotRunning.Suspended`.

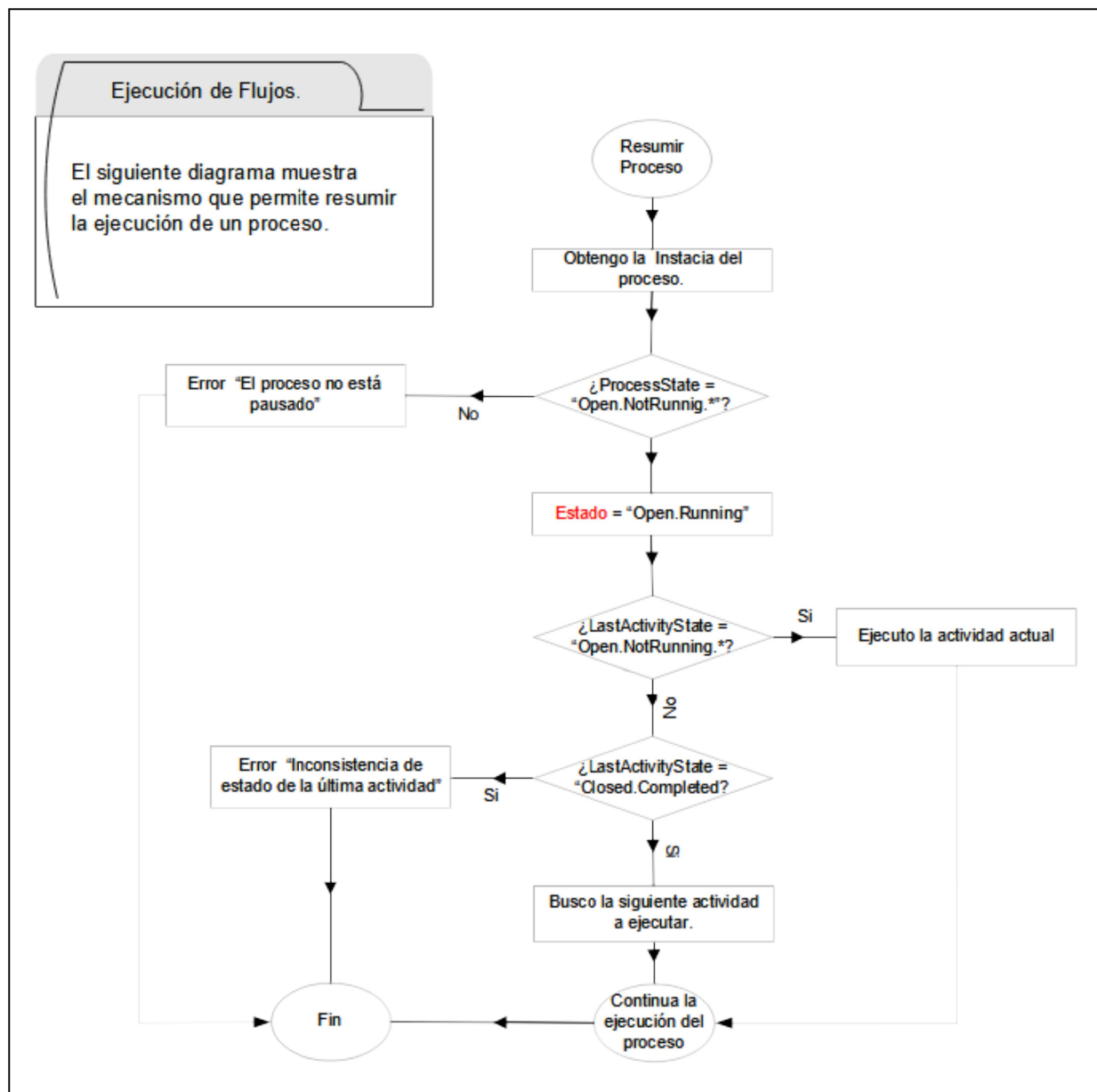
El siguiente diagrama muestra el mecanismo de pausado de un proceso.



Resume Process

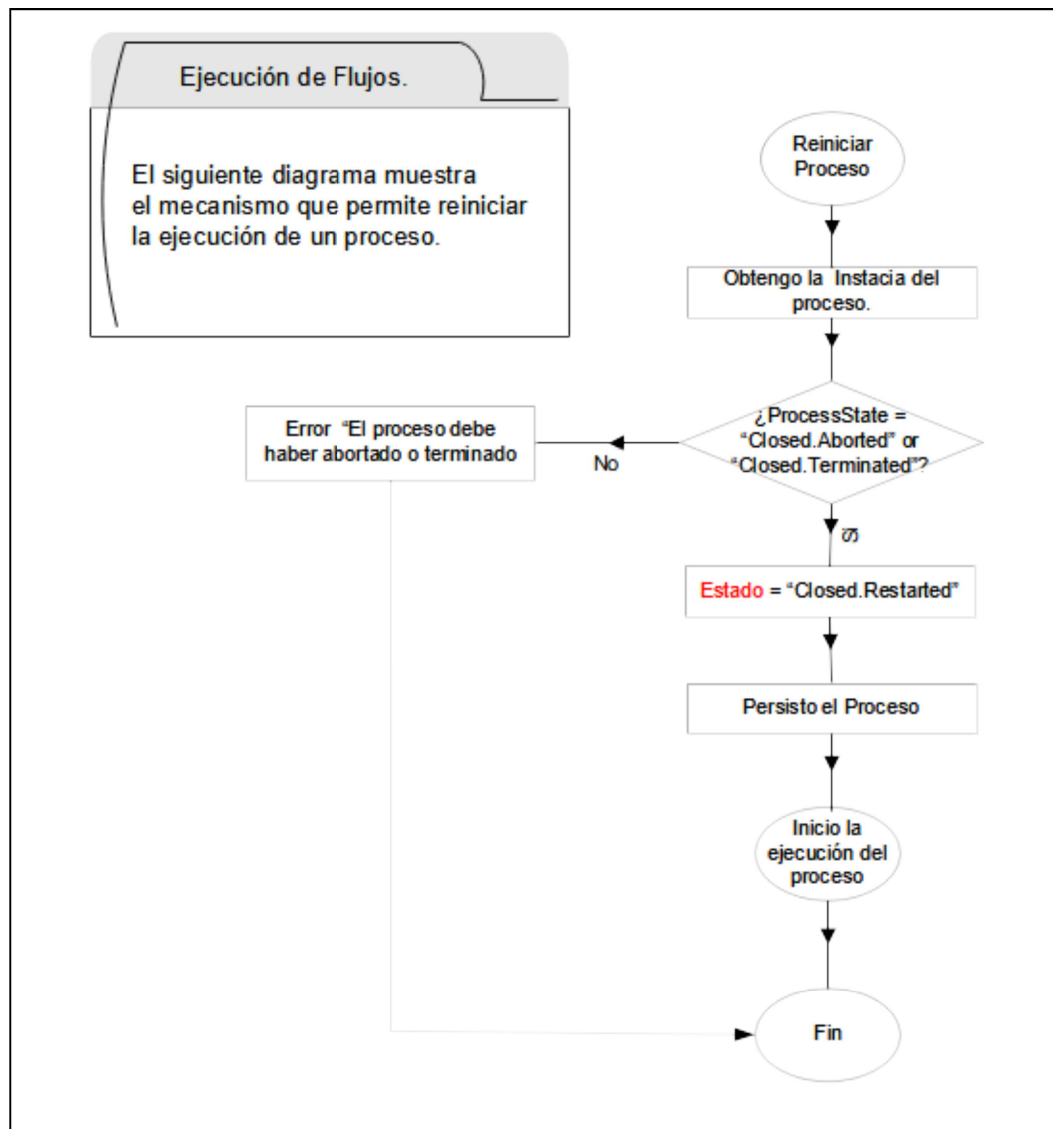
Es el método que se encarga de continuar la ejecución de un proceso que había sido pausado. En primer lugar, se obtiene la instancia del proceso, por su identificador. Si el estado del proceso es open.NotRunning.Suspended, modifica el estado a open.Running. Luego busco el estado de la última actividad ejecutada, si este es open.NotRunning se ejecuta la misma y continúa la ejecución del proceso como ya se indicó en el método Start. Si el estado es Closed.Completed, en este caso, hay que buscar la siguiente actividad a ejecutar, y luego ejecutarla.

A continuación se muestra un diagrama que indica el proceso descrito.



Restart Process.

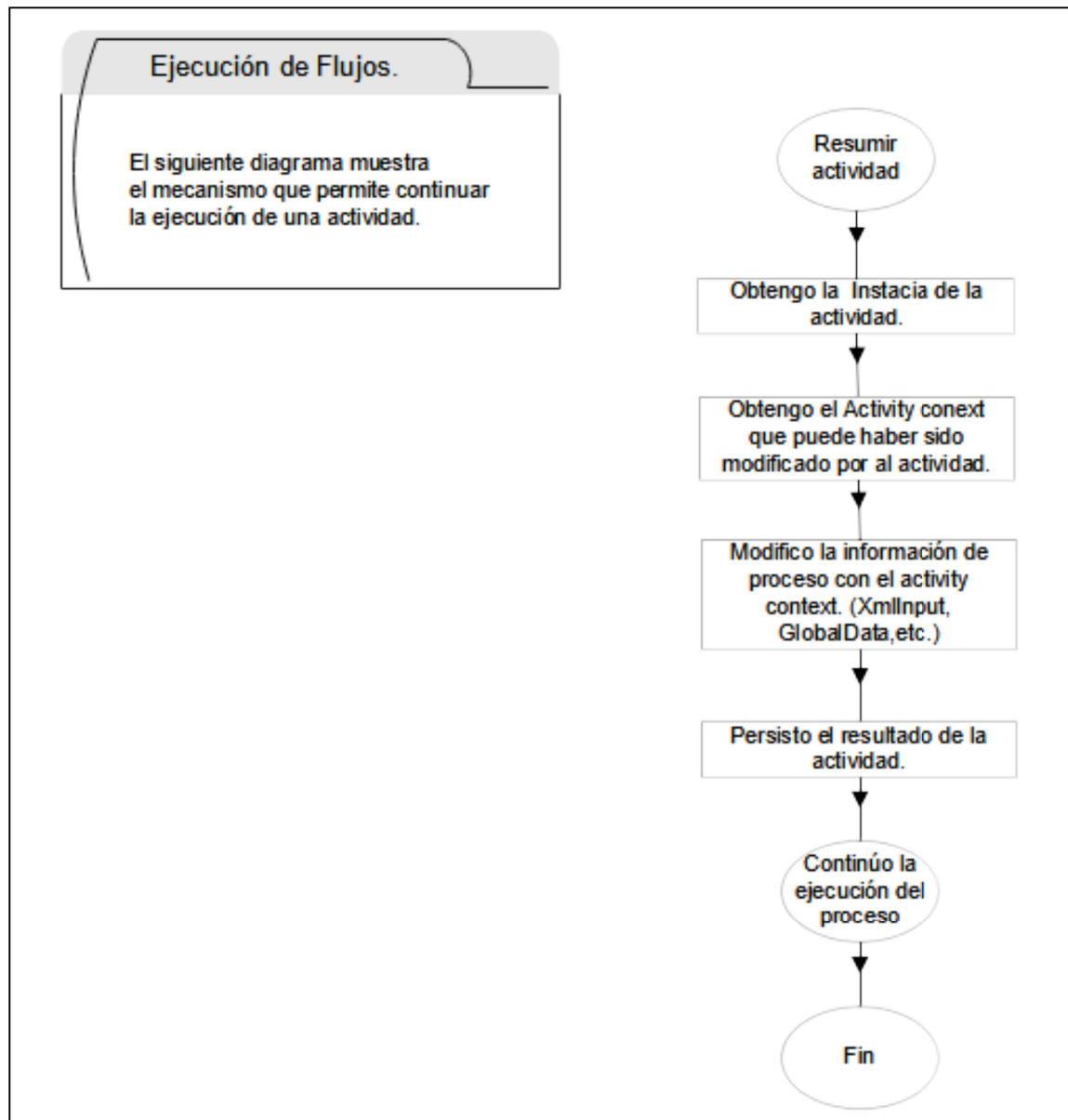
Es el método que nos permite reiniciar la ejecución de un proceso. Con tal motivo, se obtiene la instancia del proceso, por su identificador, y se procede a reiniciarlo únicamente si se encuentra en estado Closed.Aborted o Closed.Terminated. En este caso se setea el estado del proceso en Closed.Restarted y se procede a iniciar el proceso nuevamente, como se indica en el diagrama que se muestra a continuación.



Se ha señalado el comportamiento de los principales métodos del motor del workflow, observando, de esta forma, los diferentes estados por los cuales pasa. También se indicó el mecanismo de ejecución de una actividad. A continuación se describirá el mecanismo que permite resumir la ejecución de una actividad.

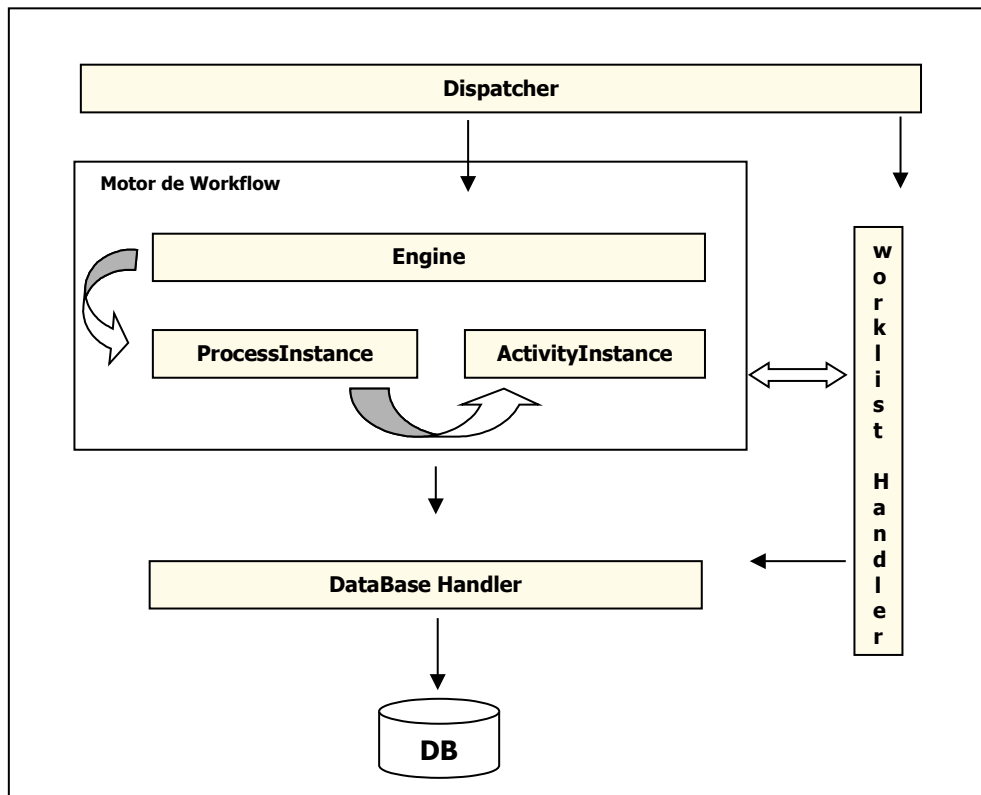
Resume activity:

Este método continúa la ejecución de una actividad que había sido pausada, típicamente utilizado en las actividades que requieren participación de un recurso humano. Cuando un proceso va a ejecutar una actividad de este tipo, automáticamente se pausa, esperando que algún recurso realice su ejecución y continúe la ejecución del proceso. En el siguiente diagrama se indica el mecanismo mediante el cual se resume un proceso.



Diseño del Motor.

A continuación se describirá el diseño del motor, indicando los principales componentes y su interacción con el resto del sistema.



Como se puede apreciar en la Figura anterior, el motor de workflow consiste de una serie de componentes que se describen a continuación.

- **ProcessInstance:** Representa las instancias de procesos que corren en el servidor. Ofrece todas las funcionalidades necesarias para poder manejar dichas instancias de procesos. O sea que cuenta con métodos necesarios para manejar las instancias como ser: un método para iniciar con la ejecución de la instancia de un proceso, un método para pausar la ejecución de la instancia, un método para la ejecución de la instancia, un método para reanudar la ejecución de la instancia que había sido pausada, etc.
- **ActivityInstance:** Representa las instancias de las actividades de un determinado proceso. Una instancia de una actividad pertenece a una instancia de un proceso (ProcessInstance). Brinda todas las funcionalidades sobre las instancias de las actividades, utilizadas principalmente por un ProcessInstance.
- **Engine:** Este componente es el que maneja las instancias de procesos. Brinda las funcionalidades anteriormente descriptas sobre las instancias de los procesos, encargándose de iniciar una nueva instancia, detenerla, reiniciarla, etc.

También se puede visualizar la interacción del motor con el manejador de listas de tareas (Worklist Handler). Dicha interacción es realizada a través de los tool Agents, invocados por las diferentes actividades.

Tanto el `ProcessInstance` como el `ActivityInstance` son independientes del motor. O sea que ni el `ProcessInstance` ni el `ActivityInstance` conocen al motor. Pero ambas clases tienen que invocar métodos que son del motor como por ejemplo el método que se encarga de la persistencia. Entonces, si ni el `ActivityInstance` ni el `ProcessInstance` conocen el motor, ¿cómo pueden invocar métodos de este?

La solución a este problema se resolvió por medio de los llamados delegates. Estos delegates no son más que los que en el lenguaje C se conocían por punteros a función. En el momento de instanciar un `ActivityInstance` o un `ProcessInstance` se le pasan delegates con los métodos que se deben invocar. Por ejemplo para el caso de la persistencia, el motor cuenta con un método que se encarga de persistir que es el método que debe usar el `ProcessInstance`, entonces en el momento de la creación del `ProcessInstance` el motor le pasa en un delegate su método para persistir. Cuando el `ProcessInstance` vaya a tener que persistir invoca el delegate para persistir que en este caso va a ser el método que se encuentra en el servidor. De esta forma se mantiene la independencia entre el `ProcessInstance` y el motor ya que el `ProcessInstance` no conoce el motor, lo único que hace es invocar el delegate que le pasan en el momento de persistir.

De esta forma, tanto el `activity` instantáneo como el `process` instantáneos, permanecerán invariados en las diferentes configuraciones del motor, de mecanismos de persistencia, etc. Siendo el motor el único "posible" afectado ante dichos cambios.

5 MANEJADOR DE LISTA DE TAREAS.

Cómo ya mencionamos el Worklist Handler, es el encargado de manejar las Worklist, por lo tanto, este debe poder manipular los diferentes Workitems que componen cada Worklist. El hecho de manejar las Worklist no incluye únicamente las funcionalidades de crear, obtener, modificar Workitems, sino que también debe permitir: reasignar un workitem, parar su ejecución, completarla, etc.

Muchas de estas funcionalidades, son utilizadas por el cliente del Workflow, por lo cual, el Dispatcher, debe interactuar con el Worklist Handler, enviándole aquellos mensajes que este debe resolver.

La Worklist, está compuesto por una cantidad de Workitems, donde, la Worklist, es la lista de tareas que un determinado recurso tiene para realizar, y un workitem representa cada una de las tareas.

Cada una de estas tareas, tiene un estado que representa su situación actual, dichos estados son:

- Open
- Open_Running
- Open_NotRunning
- Open_Suspend
- Closed
- Closed_Aborted
- Closed_Terminated
- Closed_completed.

El Worklist Handler, es uno de los componentes principales del Workflow, ya que el manejo de recursos y su asignación a las diferentes tareas, es una de las principales características de un sistema de Workflow. La asignación de tareas a recursos, puede ser realizada de forma simple, por ejemplo asignándole la tarea a todos los recursos de un determinado grupo, o bien aplicando complejos algoritmos de balanceo de carga. Pero está no es la única responsabilidad del Worklist Handler, sino que también debe notificar el estado de una determinada tarea y debe tener la habilidad de reasignar tareas a recursos, modificar su estado, pausarla, abortarla, pararla, etc. Es decir, es el encargado de proveer todas las funcionalidades sobre las Worklist, de todos los recursos, y es el único componente del sistema que va a interactuar con estas. En nuestro sistema el único de recursos que vamos a manejar son usuarios o grupos de usuarios, y las Worklist serán las tareas que un determinado usuario tiene que realizar.

Diagrama de Paquetes.

El siguiente diagrama presenta los paquetes y clases que cada uno contiene, como sus visibilidades.

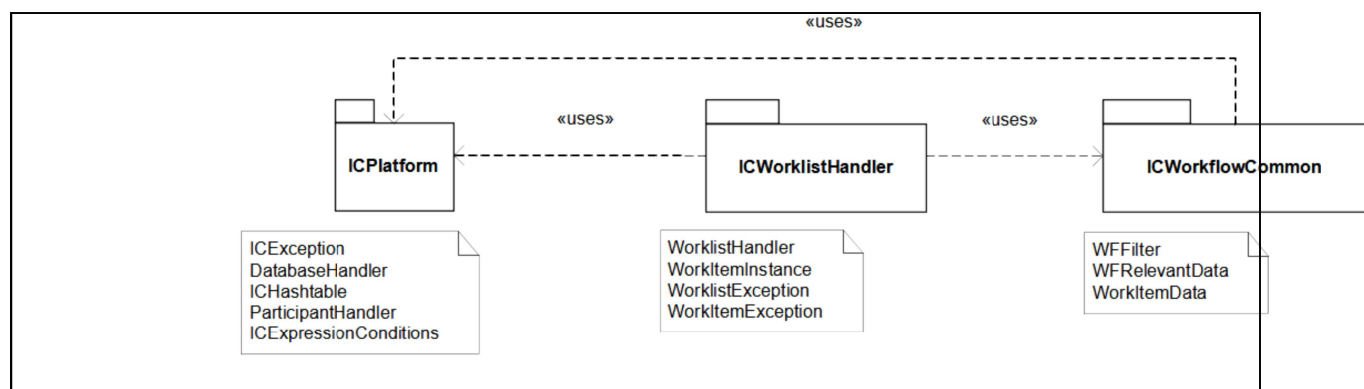
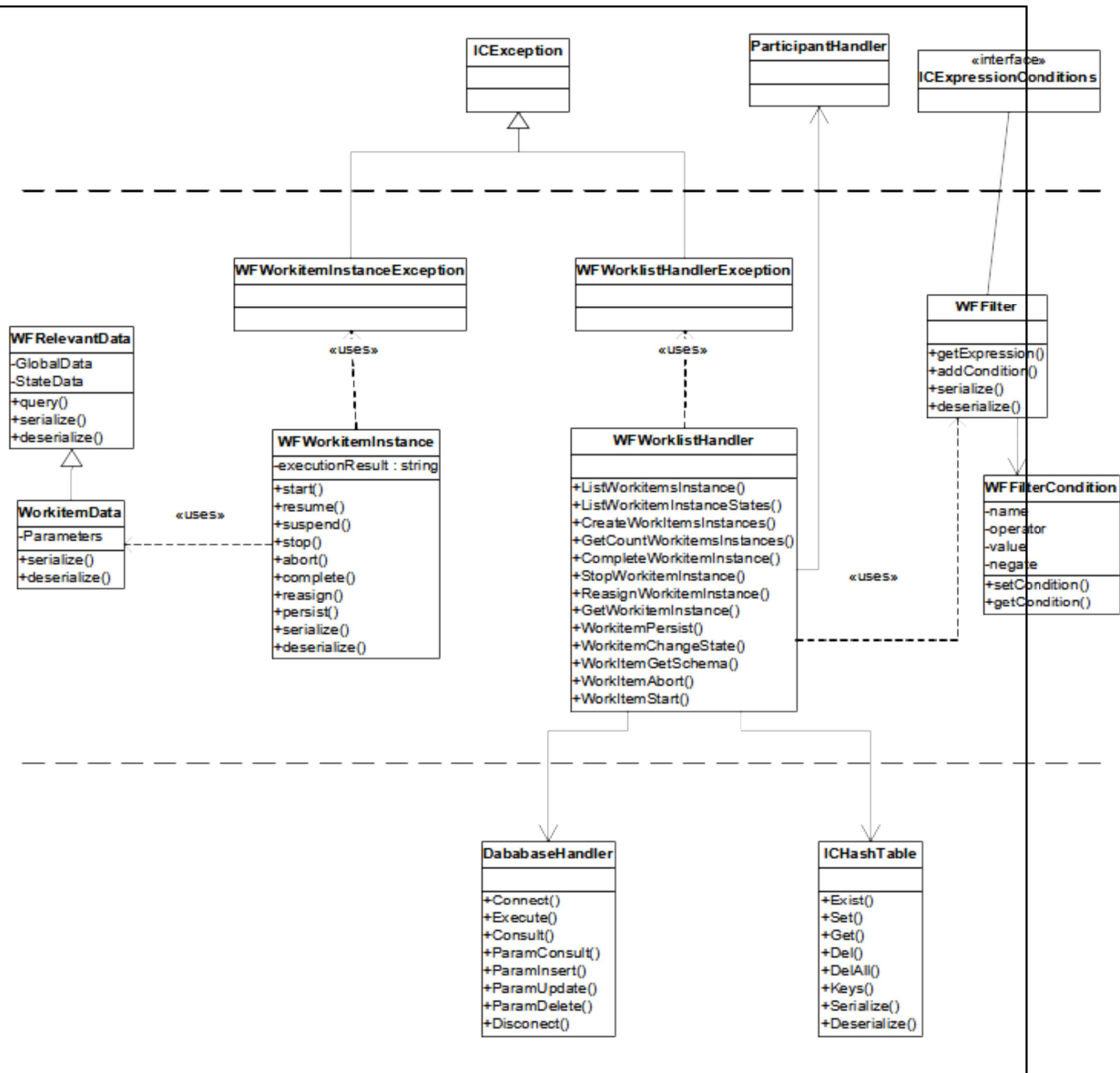


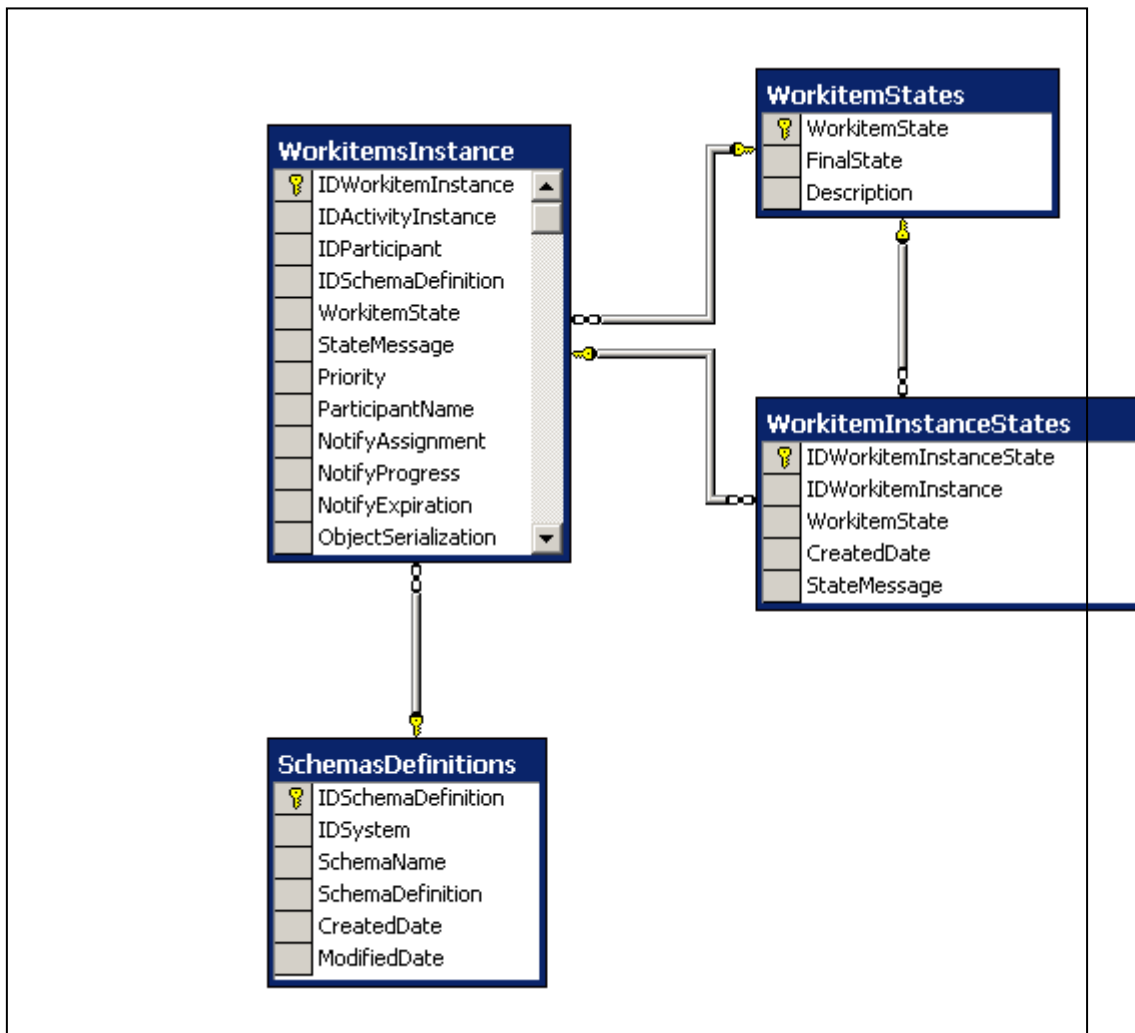
Diagrama de Clases



Como se puede visualizar en el diagrama de clases, el Worklist Handler presenta todas las funcionalidades necesarias para manipular las Worklist y los Workitems. Es importante tener en cuenta que todos los métodos del Worklist son, en realidad, delegates. Esto implica que es el Worklist Handler el que le indica cual es el método que debe ejecutarse. Por ejemplo, el workitem tiene un método que permite persistir al mismo, pero sin embargo, no tenemos visibilidad sobre el Database Handler. Esto es porque él que dice como se persiste el workitem es el Worklist Handler, quien define el método persist del workitem

Como se puede analizar en el diagrama, el Worklist Handler utiliza el Database Handler, el cual nos permite abstraernos de la forma en la que se persistirá la información. Además utilizamos el participant quien nos permite manipular los diferentes participantes que están involucrados en una determinada tarea.

Para persistir la información referente a las Worklist, Workitems utilizamos las siguientes tablas en una base de datos relacional.



WorkItemsInstance: Esta tabla almacena toda la información necesaria de cada uno de los Workitems.

WorkItemSates: Contiene los diferentes estados en los que se puede encontrar un workitem.

WorkItemInstanceStates: Indica los estados por los cuales fue pasando un workitem

SchemasDefinitions: Contiene el esquema de la información que debe ingresar el participante.

Por más información sobre las tablas, ver apéndice: Base de Datos.

6 TOOL AGENTS.

El propósito de este apéndice es el de introducir el concepto de "tool agent".

Para entender lo que son los llamados "tool agents" primero se debe de comprender el concepto de aplicación invocada, lo que la "Workflow Management Coalition" llama "Invoked Application".

Una aplicación invocada, es una aplicación que es invocada por el sistema de administración de workflow para automatizar una determinada actividad.

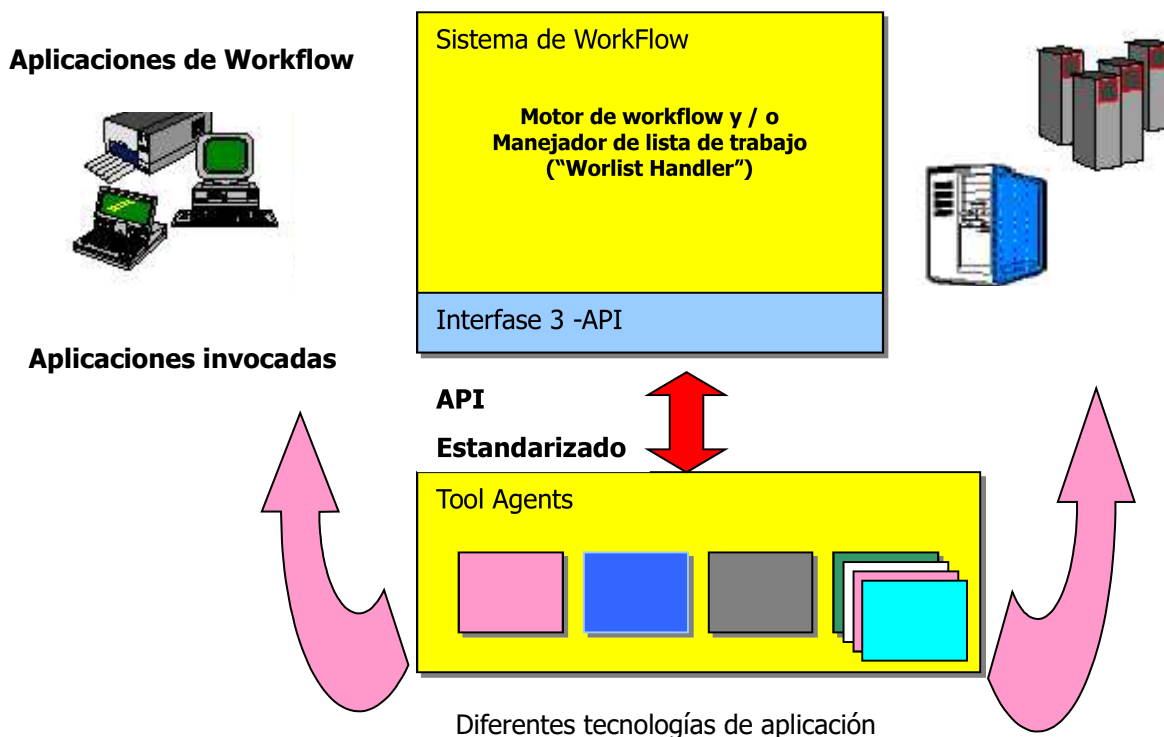
La invocación de aplicaciones puede ser una función del motor de workflow y/o del manejador de listas de trabajo ("worklist handler"). La aplicación puede ser invocada directamente por el sistema de administración de workflow o puede ser invocada indirectamente vía un agente de aplicación o también llamado "tool agent". El agente de aplicación provee un mecanismo general para la invocación de aplicaciones.

Se puede asumir que un sistema de workflow no tiene la suficiente lógica para entender como invocar todas las potenciales aplicaciones que puedan existir.

Sin embargo hay muchos sistemas de workflow que tratan con un rango de aplicaciones más restrictivas, particularmente aquellas en donde los datos son fuertemente tipados, y pueden ser asociados directamente con una herramienta particular como por ejemplo un procesador de texto.

Los agentes de aplicación o "tool agents" pueden ser vistos como manejadores o "drivers" especializados de aplicaciones.

Estos "tool agents" son los que inician y finalizan las aplicaciones externas al workflow. El sistema de workflow debe saber que "tool agents" están instalados. Existe una interfase llamada "Interfase de Invocación a Aplicaciones" que define un conjunto de funciones (API), para controlar los "tool agents". Como la interfase de aplicaciones invocadas debe manejar pedidos bi-direccionales (pedidos desde y hacia las aplicaciones), la forma de interacción con los "tool agents" depende de la interfase y de la arquitectura de las aplicaciones. Con esta interfase de invocación de aplicaciones ya no es necesario ningún mecanismo de comunicación entre los "tool agents" y el sistema de workflow. En el siguiente diagrama se puede ver gráficamente donde encaja la interfase de invocación a aplicaciones. La interfase 3 es la interfase de invocación a aplicaciones antes mencionada.



A continuación describiremos el funcionamiento de los Tool Agents, en el sistema construido. Como se señaló, la ejecución de este tipo de actividades consiste en invocar aplicaciones externas al workflow. Ejemplos de este tipo de actividades son, entre otras, la actividad que envía mail, la actividad que invoca un webservice, la actividad que ejecuta un script, etc.

Para realizar dicha ejecución se utiliza una tabla, en base de datos, llamada ToolAgents, donde se guarda para cada actividad que se implementa como Tool, cual es el toolagent que se debe usar para la invocación de la herramienta que ejecuta la actividad. Entonces lo que debe hacer el motor de workflow para ejecutar una de estas actividades es hacer una consulta en la tabla de ToolAgents y obtener cual es el toolagent asociado a la actividad a ejecutar. Luego por reflection, el motor se crea una instancia de dicho toolagent e invoca al método WMTAInvokeApplicationEx.

Todos los toolagents implementan una interfase y obligatoriamente deben implementar el método WMTAInvokeApplicationEx que es el método encargado de ejecutar la actividad. Entonces vamos a tener una clase toolagent para cada tipo de actividad, y cada una de estas clases va a contar con el método WMTAInvokeApplicationEx que es el que va a invocar el motor para ejecutar la actividad. Este método recibe un objeto de tipo WFActivityConext.

La clase WFActivityContext cuenta con los siguientes atributos:

- ProcessData;
- ActivityData;
- ActivityExecutionResult;
- SuspendExecution;

A través de la clase ProcessData se tendrá acceso a los DataFields (GlobalData), XMLInput (UserData) e información de ejecución de la instancia y sesión de usuario (SessionData).

Mediante la clase ActivityData se tendrá acceso a los atributos extendidos y parámetros de invocación (GlobalData) e información de ejecución de la instancia.

En el atributo ActivityExectutionResult se guarda el resultado de la ejecución de la actividad y finalmente el atributo SuspendExecution indica si se debe suspender la ejecución del proceso.

Vamos a ver algunos tipos de actividades a modo de ejemplo:

Actividad que envía mail

Este tipo de actividad utiliza el toolagent llamado ICToolAgentSendMail, implementando el método WMTAInvokeApplicationEx. Dicho método lo que hace es obtener: quien envía el mail ("from"), el destinatario ("to"), a quienes le llega una copia del mail ("CC"), quienes reciben copia oculta del mail ("CCO"), el asunto del mail ("Subject"), el cuerpo del mail ("Message") y cual es el servidor de salida del mail ("SMTP"). Todos estos datos son parámetros de la actividad, los saca del ActivityData del ActivityContext que recibe por parámetro. Luego utiliza una clase que con toda esta información se encarga de mandar el mail.

El código del método es el siguiente:

```
public override WFAActivityContext WMTAInvokeApplicationEx (WFAActivityContext pActivityContext)

{
    try
    {
        string IFrom = pActivityContext.ActivityData.GlobalData.Get("from");
        string ITo = pActivityContext.ActivityData.GlobalData.Get("to");
        string ICc = pActivityContext.ActivityData.GlobalData.Get("cc");
        string ICco = pActivityContext.ActivityData.GlobalData.Get("bcc");
        string ISubject = pActivityContext.ActivityData.GlobalData.Get("subject");
        string IMessage = pActivityContext.ActivityData.GlobalData.Get("message");
        string ISMTPServer=pActivityContext.ActivityData.GlobalData.Get("smtpserver");

        Infocorp.ICPlatform.ICUtilities.MailUtil.SendMail(IFrom, ITo, ICc, ICco, ISubject, IMessage, null,
ISMTPServer);
    }

    catch(Exception e)
    {
        pActivityContext.ActivityExecutionResult = "Ha ocurrido el siguiente error: " + e.Message;
    }
    pActivityContext.SuspendExecution = false;
    return pActivityContext;
}
```

Actividad que ejecuta un script

Este tipo de actividad tiene asociado el toolagent llamado ICToolAgentScriptCall, en este caso, el método WMTAInvokeApplicationEx lo que hace es obtener el lenguaje que es el utilizado para el script, (puede ser VBScript o Jscript) y obtiene también el código del script, tanto el lenguaje como el código del script vienen dentro del ActivityData que viene en el ActivityContext que le pasan por parámetro al método. Una vez obtenido tanto el lenguaje como el código del script, se ejecuta el script utilizando un control de .NET que sirve para ejecutar scripts.

El script tiene que tener una determinada estructura, debe tener un método main el cual recibe como parámetro un xml que es el ActivityContext serializado, este método main es el que ejecuta el control de script.

```
public WFAActivityContext WMTAInvokeApplicationEx (WFAActivityContext pActivityContext)
{
    ScriptControl IScript = new ScriptControl();
    WFAActivityContext IResult;

    if (! pActivityContext.ActivityData.GlobalData.Exists("sourcecodetype"))
    {
        pActivityContext.ActivityData.GlobalData.Set("sourcecodetype", "VBScript");
    }
    IScript.Language = pActivityContext.ActivityData.GlobalData.Get("sourcecodetype");
    try
    {
        IScript.AddCode((string)(pActivityContext.ActivityData.GlobalData.Get("source    code")));
    }
    catch(Exception e)
    {
        throw new ICToolAgentScriptCallException("El script no es válido.", e);
    }
    pActivityContext.ActivityData.GlobalData.Del("sourcecodetype");
    pActivityContext.ActivityData.GlobalData.Del("sourcecode");
    try
    {
        //La invocación al método Run deberá devolver un string, el cual será la //serialización de un objeto
        ProcessData
        object[] IParams = {pActivityContext.serialize()};
        object IScriptResult = IScript.Run("main", ref IParams);
        IResult = WFAActivityContext.deserialize((string)IScriptResult);
    }
    catch(Exception e)
    {
        throw new ICToolAgentScriptCallException("Ha ocurrido un error en la ejecución del script.",
        e);
    }

    return IResult;
}
```

7 COMPONENTES REUSABLES.

En este apéndice se detalla en profundidad, aquellos componentes, utilizados por el sistema de WorkFlow, y que fueron pensados para ser reusados en otros proyectos. Entre los cuales se incluye:

- Gestor genérico de participantes.
- Gestor de sesiones.
- Generador y Evaluador de expresiones.
- Manejador de Base de Datos

7.1 Gestor genérico de participantes.

Introducción

Como ya se ha discutido en otros estudios, una de las principales funcionalidades que debe brindar un workflow es la de asignar recursos a diferentes actividades. La forma en la que se realizará dicha asignación esta fuera del alcance de este documento, ya que se puede manejar un amplio espectro de posibilidades, desde asignar el primer recurso disponible a dicha actividad hasta complejos algoritmos de balanceo de carga.

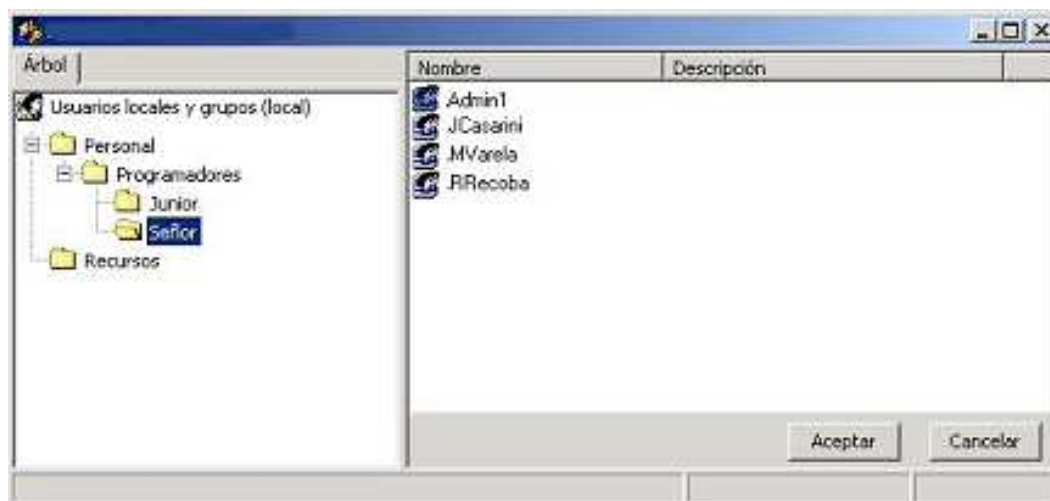
En este documento nos centraremos en diseñar una arquitectura adecuada para la definición de participantes (recursos). Comenzaremos estudiando que componentes del workflow necesitan conocer los participantes que actúan en el sistema y que información necesitan, en base a esto, definiremos los diferentes componentes y los mecanismos de comunicación.

Incidencia de los Participants en el sistema.

En primera instancia los participants deben poder ser accedidos desde el editor de flujos, ya que es aquí donde se asignarán a las diferentes actividades que denominemos "UserTask", es decir, aquellas tareas que requieran un recurso para ser realizadas. Para poder seleccionar recursos, se realizará un UserControl, que permita visualizar y seleccionar recursos.

Por esto, necesitamos funcionalidades que nos permitan obtener los distintos grupos existentes, los posibles subgrupos que lo integren, y todos sus usuarios, estos son, los usuarios del grupo y todos los usuarios de todos los subgrupos que este contenga.

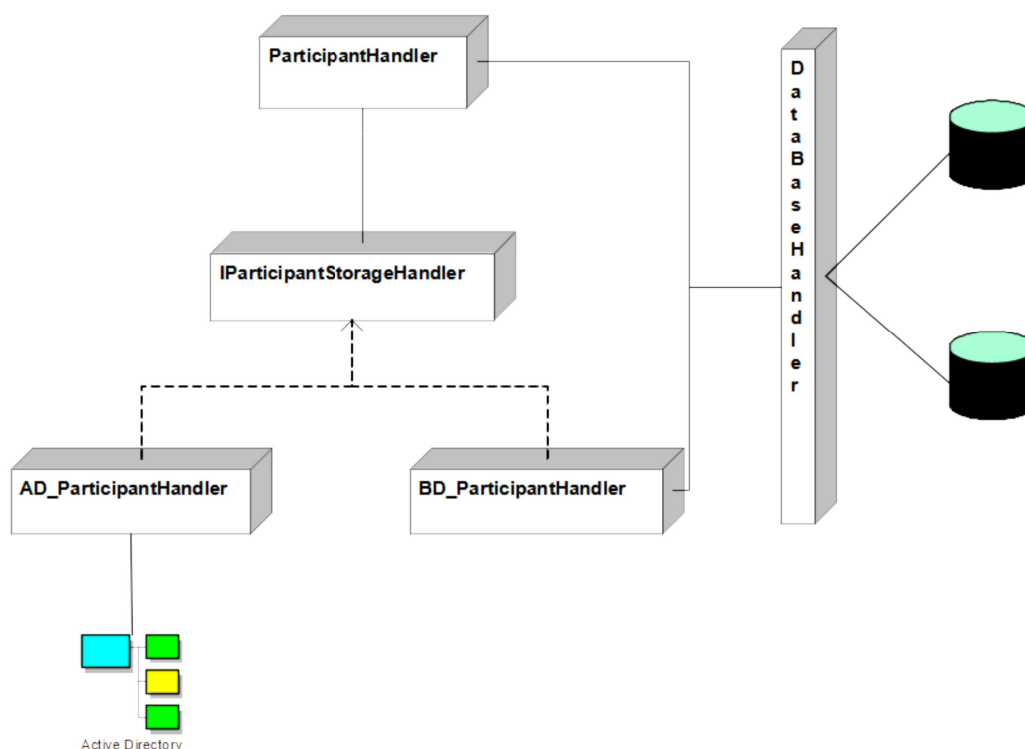
Es importante tener en cuenta que un cliente ya puede tener la información de usuarios y recursos en una base de datos propia, o en Active Directory o en cualquier otra fuente de datos, por lo tanto el diseño que se presentará deberá tener en cuenta dichas posibilidades, y a su vez, brindar una alternativa para aquellos clientes que no tengan dicha información.



Además del editor de flujos, el workflow enactment service también necesitara información de los participantes, ya que mediante el worklistHandler deberá encargarse de notificar al recurso involucrado de dicha tarea, por lo tanto necesitara toda la información del recurso como ser: mail, disponibilidad, cargo, etc.

Diseño.

En la siguiente imagen mostramos el diseño para el manejo de participants y a continuación detallaremos cada uno de sus componentes



ParticipantHandler: Este componente será utilizado desde el editor de flujos, siendo el encargado de proveernos la información que necesitemos sobre los grupos y recursos de cada grupo. A su vez, será utilizado por el workflow enactment service quien lo utilizará para extraer la información necesaria de cada recurso, así como también, tendrá la habilidad de realizar altas, bajas y modificaciones. Por lo tanto dicho componente debe poseer las habilidades de consulta y ABM.

Es probable que los clientes ya tengan almacenada la información de sus recursos en diferentes sistemas de información, por lo cual se deberá crear componentes que nos permitan acceder a los diferentes "storages", es por esto que manejamos el concepto de Type. Cada participante pertenecerá a un determinado type dependiendo del componente encargado de acceder a storage que necesite.

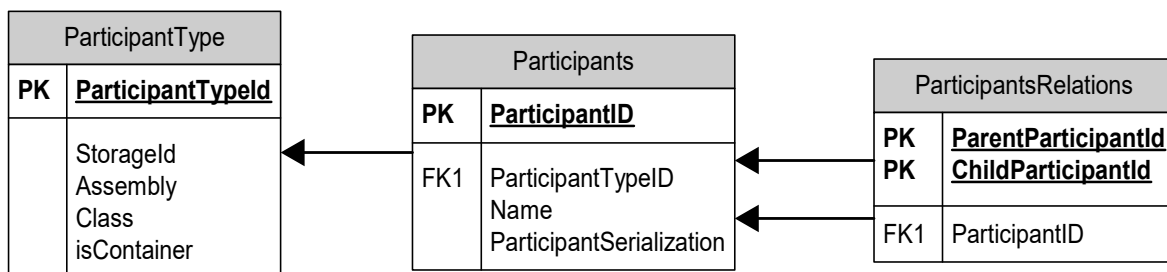
IParticipantHandler: Para facilitar el desarrollo de los componentes, se especificará la interfaz que deben implementar los diferentes XX_ParticipantHandler, de esta forma, alcanza con implementar dicha interfaz, con la habilidad de acceder al storage correspondiente, y agregar un nuevo tipo en la siguiente tabla:

De esta forma, el ParticipantHandler simplemente tendrá que consultar la tabla, para saber que componente será el encargado de acceder al storage.

StorageTypes	
PK	<u>IdStorage</u>
	Name Assembly StorageClass Image

Manejo de usuarios:

Para el manejo de usuarios, se brindaran las siguientes tablas que nos permitirán manejar participantes y relaciones entre participantes, ya que un participante puede ser un conjunto de participantes (container).

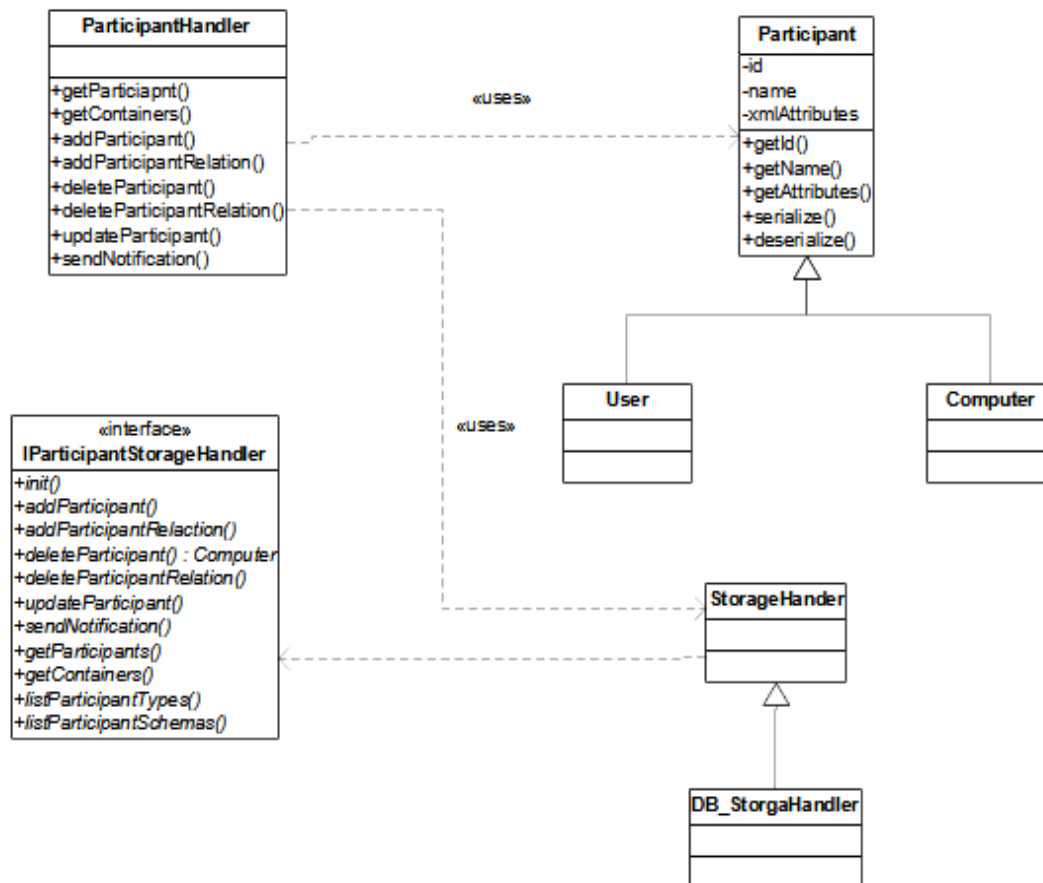


Como los participantes pueden ser usuarios, computadoras, recursos, etc., la tabla de participantes contiene un campo info, que mantendrá la información necesaria para ese participante en formato xml. Cada participante tendrá un tipo donde se indica entre otras cosas el assembly y clase que lo implementa, el mismo poseerá un método que le permitirá serializarse a partir de los datos almacenados en la base de datos.

Además se implementará una BD_ParticipantHandler para el manejo de usuarios, utilizando las tablas antes definidas.

Diagrama de Clases.

A continuación se muestra un diagrama de clases. Como se puede ver, el Participant Handler brinda todas las funcionalidades para la gestión de participantes, mediante el uso de una interfaz IParticipantStorage Handler la cual es la encargada de obtener la información, según sea necesario.



7.2 Manejador de sesiones.

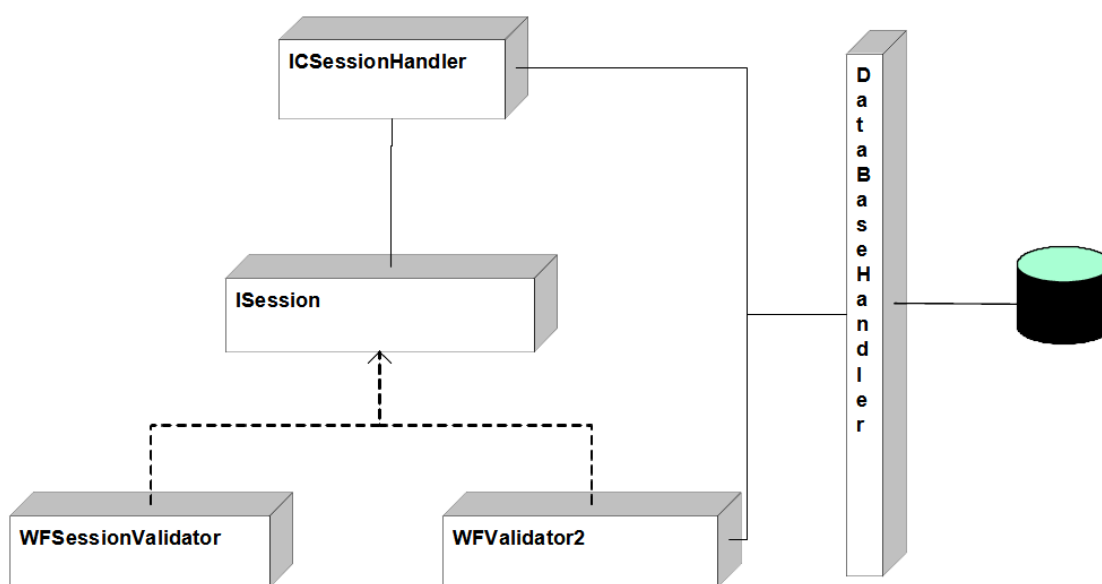
Introducción.

El sistema de Workflow debe manejar usuarios, permisos, perfiles, etc. Es por esto que se manejará el concepto de sesión de usuario.

Para ingresar al sistema, el usuario debe ingresar su nombre de usuario y contraseña, esta información es validada y según su perfil podrá acceder a diferentes facilidades dentro del sistema. La sesión, posee, entre otras propiedades, el perfil, la fecha de creación, fecha de último acceso, etc.

Diseño.

En la siguiente imagen mostramos el diseño para el manejo de sesiones y detallaremos cada uno de sus componentes.

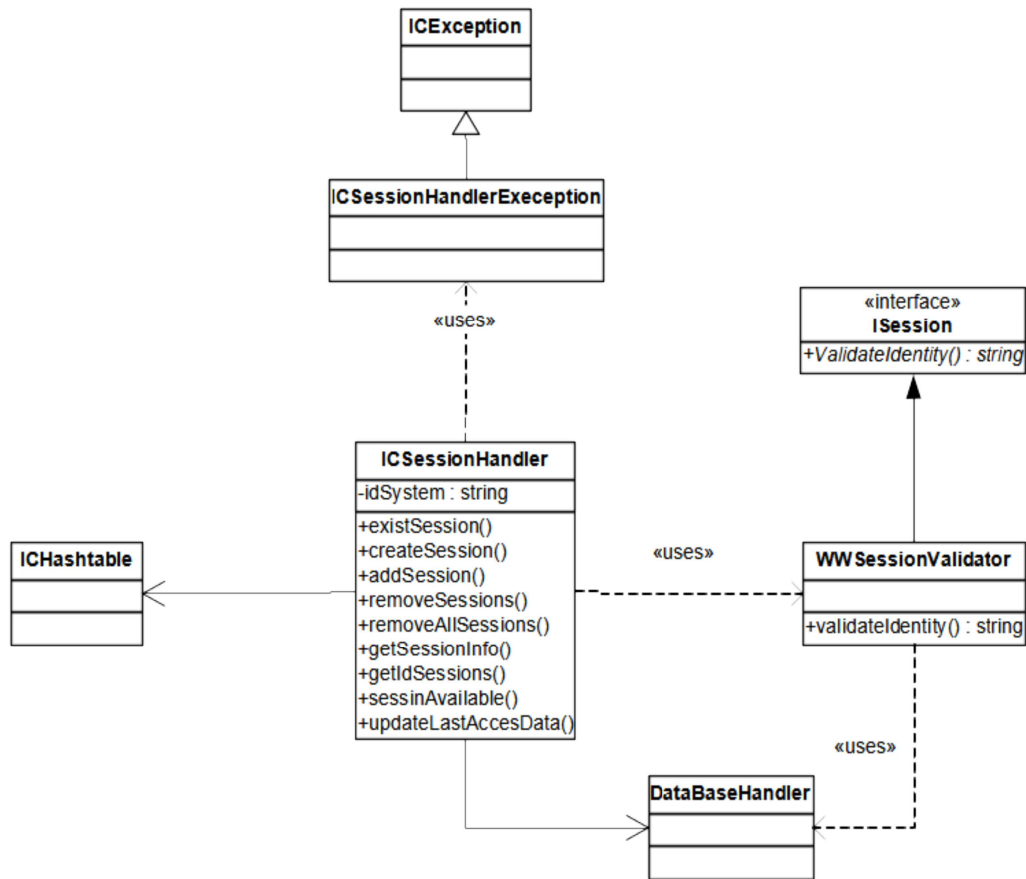


ISessionHandler: Este componente será el encargado de manejar, crear y eliminar las sesiones de usuario. Este componente utiliza otros que implementen la interfaz ISession, de esta forma, para cada sistema tendremos un componente que nos permita validar una sesión, según los requisitos que este contemple.

ISession: Para facilitar el desarrollo de los componentes, se especificará la interfaz que deben implementar los diferentes validadores, de esta forma, alcanza con implementar dicha interfaz, con la habilidad de validar usuarios, e indicar en una tabla cual es el componente encargado de validar usuarios al mismo. Dicha tabla debe tener, el assembly y la clase encargada de validar las sesiones, con tal motivo, se utilizó la tabla de Sistemas, ya que un sistema utiliza un único validador.

De esta forma, el Session Handler simplemente tendrá que consultar la tabla, para saber que componente será el encargado de validar los usuarios del sistema.

Diagrama de clases



7.3 Generador y Evaluador de expresiones

Construcción de expresiones

Se implementó un namespace llamado ICExpressionBuilder el cual contiene un conjunto de clases que se utilizan para la construcción de expresiones en un determinado lenguaje.

La idea es brindar la posibilidad de construir expresiones en el lenguaje que se quiera, por ejemplo la expresión "a=3 O b=4", se puede escribir de varias formas. Si se quiere la expresión anterior en SQL, quedaría "a=3 OR b=4", en C# se escribiría "a==3 || b==4", para cada lenguaje la expresión anterior se escribe de una forma distinta.

El namespace trae clases para poder construir expresiones tanto en C# como en SQL pero se deja abierta la posibilidad de construir expresiones en el lenguaje que se desee. Para esto se crearon una serie de clases abstractas e interfaces para que el desarrollador si lo desea pueda crear las expresiones en el lenguaje que desee, simplemente implementando las clases adecuadas.

Diseño

La clase que se utiliza para construir expresiones se llama "Expression". Para crearnos una expresión lo que se debe hacer es instanciar un objeto del tipo Expression y cargarle tanto el elemento izquierdo, el derecho y el operador que forman parte de la expresión. Tanto el elemento izquierdo, como el derecho, son objetos del tipo ExpressionElement. Para generar el string con la expresión se utiliza el método llamado buildExpression sobre el objeto que se instanció antes, pasándole por parámetro la clase que se utiliza para escribir la expresión en un determinado lenguaje. Por ejemplo se quiere que se genere un string con la expresión en lenguaje SQL se le debe pasar por parámetro al método buildExpression la clase ExpressionOperatorSQL. Para clarificar más los conceptos en el siguiente ejemplo se puede ver como se construye la expresión "a=3" en lenguaje C#:

```
Expression IExpression = new Expression(new ExpressionElement(
    new ExpressionValue("a"), ExpressionOperators.EQUAL,
    new Expression(new ExpressionElement(new ExpressionValue("3"))

string ICondition = IExpression.buildExpression(new ExpressionOperatorCS());
```

La clase ExpressionOperatorCS es la que utiliza el buildExpression para construir la expresión anterior en lenguaje C#. Si en el ejemplo anterior hubiéramos querido generar una expresión en SQL se debería invocar al método buildExpression de la siguiente forma:

```
ICondition = IExpression.buildExpression(new ExpressionOperatorSQL());
```

Si quisiera generar la expresión en otro lenguaje que no sea ni SQL ni C#, lo que se debe hacer es implementar una nueva clase. En el siguiente diagrama se puede apreciar como se debería implementar esta nueva clase para generar una expresión en un determinado lenguaje.

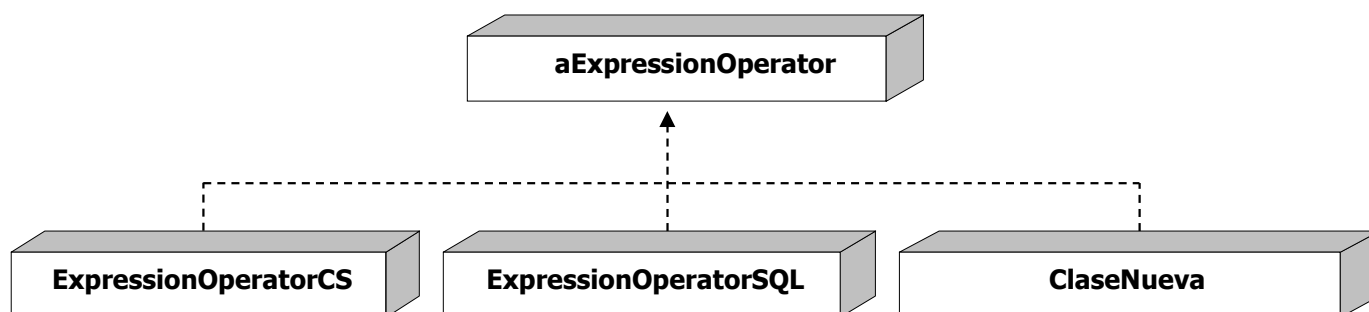


Figura 1

Como se puede apreciar en la figura anterior la "ClaseNueva" debe heredar de la clase abstracta llamada `aExpressionOperator`. En particular va a tener que implementar el método `getValue()` que es el método en el cual se traducen los operadores a un determinado lenguaje.

Por ejemplo en el caso de la clase `ExpressionOperatorCS` lo que hace el método `getValue()` es traducir el símbolo de igualdad a `==`, la condición de y a `&&`, la condición de "o" a `||`, etc.

Evaluación de Expresiones

Se implementó un namespace llamado `ICCodeEvaluator` en el cual se encuentran las clases para evaluar las expresiones generadas por el namespace `ICBuildExpression`. La clase dentro del namespace que se utiliza para evaluar las expresiones se llama `CodeEvaluator`, y los métodos que brinda para evaluar las expresiones son:

- `int EvaluteToInt(string pExpression)` Evalúa la expresión retornando un entero.
- `string EvaluteToString(string pExpression)` Evalúa la expresión retornando un string.
- `bool EvaluteToBool(string pExpression)` Evalúa la expresión retornando un booleano.
- `bool EvaluteToObject(string pExpression)` Evalúa la expresión retornando un objeto en el resultado de la misma.

Para el caso de que la expresión que se haya generado sea en C#, la forma de evaluarla es generar código C# en tiempo de ejecución, compilar dicho código en memoria, ejecutarlo y devolver el resultado. El código generado en memoria va a ser la expresión que se quiere evaluar. En el siguiente ejemplo se puede apreciar como se evalúa una expresión que se generó en C# utilizando el namespace para la construcción de expresiones llamado `ICBuildExpression`

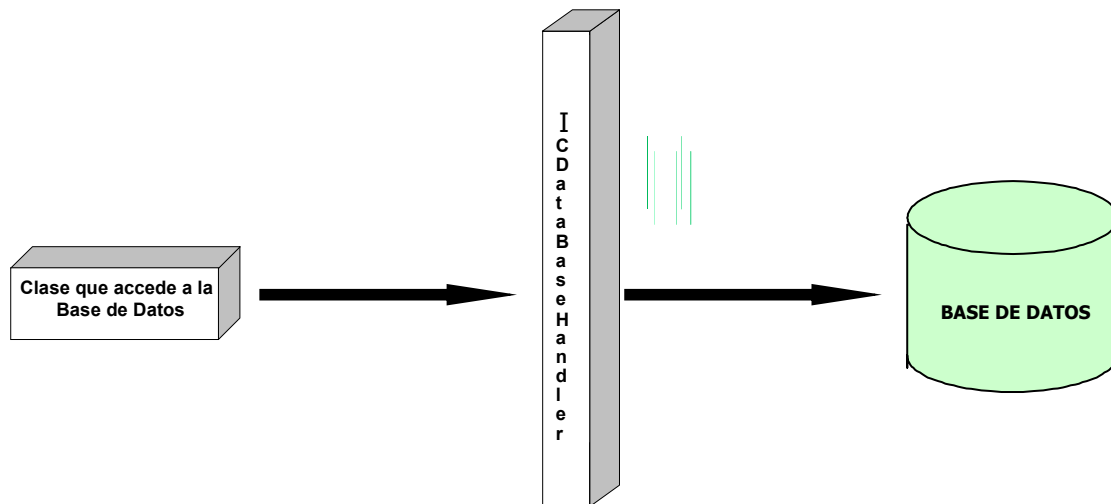
```
Expression IExpression = new Expression(new ExpressionElement(  
new ExpressionValue("a"), ExpressionOperators.EQUAL,  
new Expression(new ExpressionElement(new ExpressionValue("3")  
string ICondition = IExpression.buildExpression(new ExpressionOperatorCS());  
bool IResultado = CodeEvaluator.EvaluateToBool(ICondition);
```

En este caso se utilizó el método `EvaluateToBool` porque lo que se evaluaba era una condición y el resultado es true o false.

7.4 Manejador de Base de Datos

ICDataBaseHandler es un namespace que brinda una capa intermedia para el acceso a los datos. Dentro del namespace ICDataBaseHandler se encuentran las clases con los métodos para interactuar con las bases de datos.

La idea es que las clases que quieran acceder a una determinada base de datos lo hagan a través de la ICDataBaseHandler. O sea que ICDataBaseHandler va a ser como un intermediario para acceder a las bases de datos.



Dentro del namespace ICDataBaseHandler se encuentra la interfase IDatabaseHandler. Dicha interfase brinda los métodos necesarios para acceder a una determinada base de datos. Toda clase que quiera tener acceso a datos que se encuentran en una base de datos debe usar los métodos que ofrece la clase IDatabaseHandler.

La ventaja de tener esta capa intermedia de acceso a los datos es que la clase que quiere interactuar con la base de datos no se debe preocupar de como implementar dicha interacción, simplemente invoca los métodos de la clase IDatabaseHandler.

Métodos para el acceso a los datos

Los métodos que brinda la interfase IDataBaseHandler para acceder a los datos son los siguientes:

- **Connect** : Recibe un string de conexión y se conecta a la base de datos especificada en el string de conexión con el usuario y el password también especificado en dicho string de conexión.
- **Consult** : Ejecuta una consulta SQL que recibe como parámetro.
Dicha consulta es del tipo SELECT – FROM –WHERE.
Este método devuelve un dataset cargado con el resultado de la consulta.
- **Execute** : Ejecuta una sentencia SQL que recibe como parámetro.
La sentencia a ejecutar es una sentencia SQL que no devuelve nada o sea no es una consulta del tipo SELECT sino que es una sentencia del tipo INSERT, DELETE, etc.
- **ParamConsult** : Ejecuta una consulta parametrizada SQL que recibe como parámetro.
Este método recibe como parámetro la consulta parametrizada y los parámetros de dicha consulta y devuelve un dataset cargado con el resultado de la consulta.

Un ejemplo de consulta parametrizada es:

```
SELECT * FROM TABLA WHERE ID=? (? = parámetro)
```

- **ParamInsert** : Ejecuta una sentencia SQL Insert parametrizada que recibe como parámetro.
Un ejemplo de un Insert parametrizado:

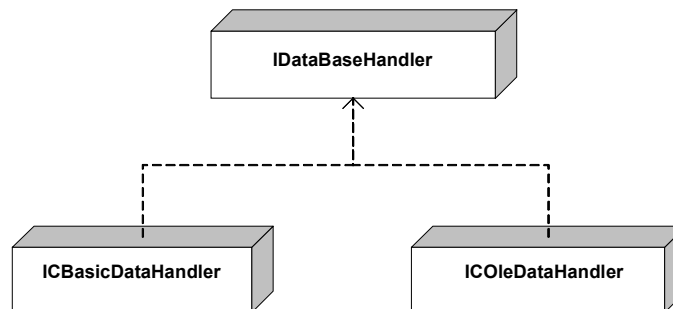
```
INSERT INTO TABLA VALUES(?,?) (? = parámetro)
```

- **ParamDelete** : Ejecuta una sentencia SQL Delete parametrizada que recibe como parámetro.
- **ParamUpDate** : Ejecuta una sentencia SQL Update parametrizada que recibe como parámetro.
- **Disconnect** : Ese método hace que se produzca la desconexión de la base de datos.

Diseño

Como ya se mencionó la interfase `IDataBaseHandler` ofrece los métodos para el acceso a datos. Dicha interfase es implementada por 2 clases, `ICBasicDataHandler` e `ICOLEDataHandler`. Cada una de estas clases brinda una implementación distinta para los métodos de acceso a los datos.

En la clase `ICBasicDataHandler` se usa `SqlConnection`, mientras que en la clase `ICOLEDataHandler` se utiliza `OleDbConnection` para interactuar con las bases de datos. Si en algún momento se quiere tener alguna otra implementación de los métodos de la interfase `IDataBaseHandler` simplemente se hace una nueva clase que implemente dicha interfase.



7.5 Lenguaje para referenciar información del proceso.

Introducción

Se debe proveer al desarrollador encargado de definir el flujo de un proceso, un lenguaje por medio del cual pueda acceder a datos del formulario de ingreso y a variables globales en el momento de definir el flujo de un proceso. Es muy importante que dicho desarrollador pueda referenciar estos datos en el momento de definir un flujo, para que las actividades que este defina, puedan referenciar tanto a datos ingresados en el formulario como a variables globales.

Para comprender mejor la necesidad de este lenguaje vamos a ver un breve ejemplo:

Supongamos que un desarrollador tiene que definir el flujo de un proceso en el cual los empleados hacen una cierta solicitud en una empresa. Para esto al proceso se le asocia un formulario de entrada por el cual el empleado que hace la solicitud ingresa sus datos. Entre otros, dicho empleado debe completar su dirección de mail. Mas adelante, este proceso tiene una actividad de mail que manda cierta notificación al empleado que hizo la solicitud. Le debe mandar la notificación a la dirección de mail ingresada por el empleado en el formulario de ingreso del proceso. Para esto cuando se crea la actividad de mail que manda la notificación al empleado, se debe poder ingresar de alguna forma que el destinatario del mail sea la dirección de mail ingresada. En la siguiente figura, se muestra como quedaría el formulario asociado a esta actividad de mail.

Propiedades: Send Mail - Microsoft Internet Explorer

Paso 2 de 2
Ingrese aquí las propiedades asociadas a la actividad.

Título:	Notificacion
Descripción:	Notifica al usuario el resultado de su solicitud.
Servidor SMTP:	Apolo
De:	IC-Admin@ICorp.com.uy
Para:	{StateData.XmlInput.//Mail}
CC:	
CCO:	
Asunto:	Solicitud de Licencia
Mensaje:	Resultado Aprobacion: {StateData.XmlInput.//Descripcion1} Resultado Verificacion: {StateData.XmlInput.//Descripcion2}

Anterior Aceptar Cancelar

Figura 1

Como se puede apreciar en la figura anterior en el campo llamado "Para" se ingresa la siguiente expresión:

{StateData.XmlInput.//Mail}

Esta expresión está escrita en un lenguaje, que se explicará más adelante, y referencia al campo llamado "mail" del formulario de ingreso de datos, en donde el empleado ingresó su dirección de mail.

También se puede apreciar que dentro del mensaje del mail también se utiliza este lenguaje para obtener los valores de otros campos del formulario de ingreso de datos.

Descripción del lenguaje utilizado

Se verá primero como es la sintaxis de las expresiones que quieran referenciar a variables globales. Las expresiones para poder referenciar dichas variables globales al proceso deben tener la siguiente forma:

"GlobalData.NombreVariableGlobal"

En donde GlobalData indica que se va a hacer referencia a una variable global, y NombreVariableGlobal se debe sustituir por el nombre de la variable global que se quiere referenciar.

Por ejemplo, si se define una variable global de tipo integer, llamada "VarGlobal" con un valor inicial de 0, para poder ser referenciada desde una determinada actividad, debe escribir la siguiente expresión:

"GlobalData.VarGlobal"

En el momento de procesar esta actividad, el motor de WorkFlow sustituye esta expresión por su respectivo valor que en este caso es 0.

Para poder acceder a datos del formulario de ingreso, la sintaxis de las expresiones debe tener la siguiente forma:

StateData.XmlInput.//Campo

En donde StateData.XmlInput indica que se va a referenciar un campo del formulario de ingreso, y Campo es el nombre del campo dentro del formulario de ingreso al cual se hace referencia.

Por ejemplo, si dentro de una actividad se quiere referenciar un campo llamado mail del formulario de ingreso de datos, se debe escribir la siguiente expresión:

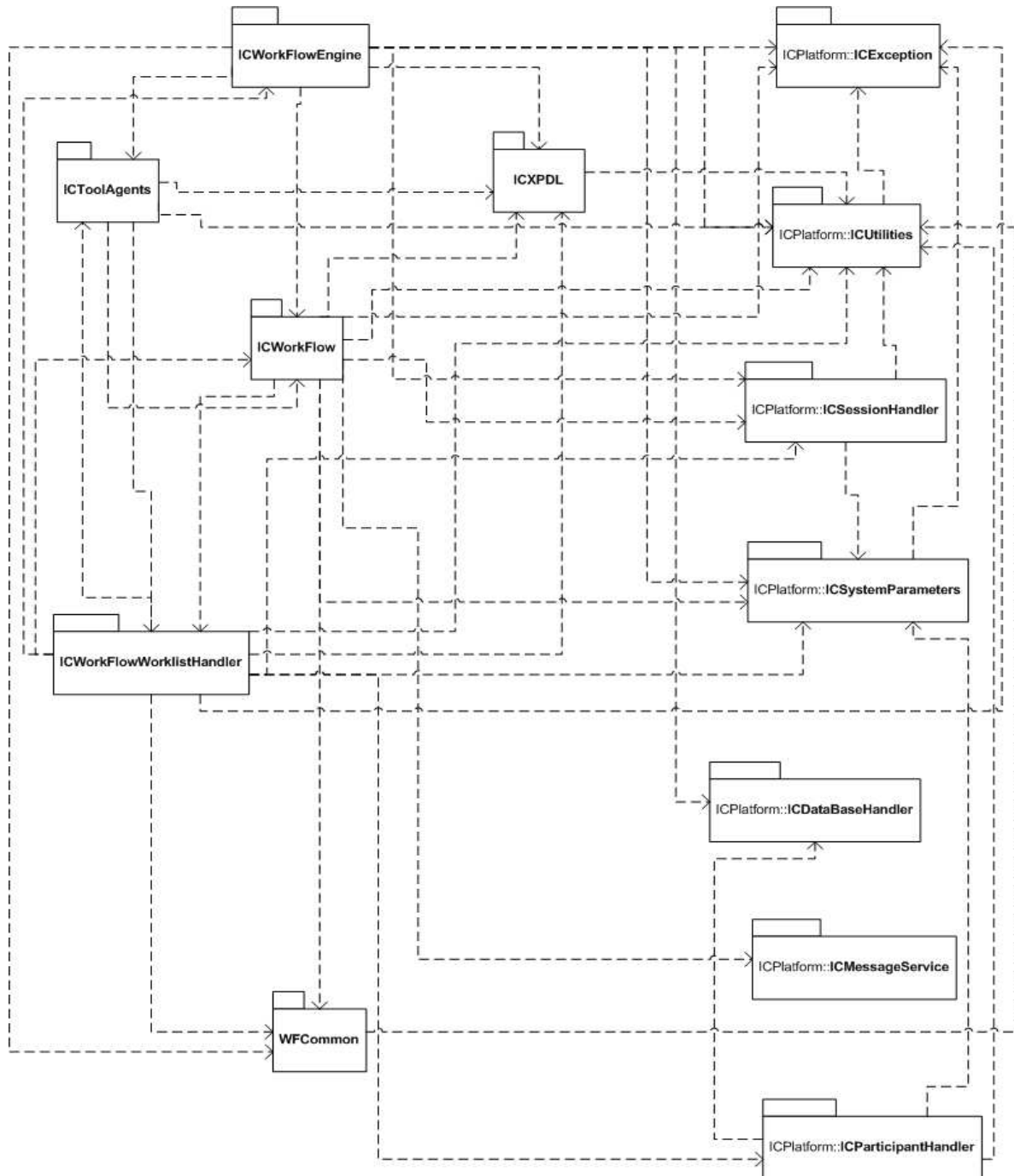
StateData.XmlInput.//mail

De esta forma, cuando el motor del WorkFlow procese esta actividad, va a sustituir esta expresión por el valor correspondiente en el formulario de ingreso de datos.

8 DISEÑO DE CLASES, PAQUETES, DEPLOY.

Diagrama de paquetes

En la siguiente Figura, se muestra un diagrama de paquetes, mostrando los paquetes del sistema y sus dependencias.



Se verá a continuación una descripción de cada tabla de la base de datos.

Tablas para manejo de procesos y actividades

Tabla ProcessesDefinition

Esta es la tabla se utiliza para almacenar las definiciones de procesos creadas. Cuando se define un nuevo proceso se crea un nuevo registro en esta tabla en donde se almacenan los datos del mismo. El motor es quien interactúa con esta tabla para almacenar, consultar, modificar, eliminar, obtener y listar definiciones de procesos.

La estructura de dicha tabla es la siguiente.

Campo	Tipo	Observaciones
IDProcessDefinition	Char	Identificación de la definición de proceso.
IDSysSystem	Char	Nombre del sistema al cual pertenece la definición de proceso.
Name	Varchar	Nombre del proceso.
IDSchemaDefinition	Char	Identificador del schema que se utilizará para validar la entrada.
ObjectSerialization	Text	Serialización del objeto
Enabled	Bit	Indica si se puede crear una instancia de proceso. Cuando un proceso se marca como eliminado, este valor se pone automáticamente en false.
Transactional	Bit	Indica si el proceso es Transaccional
Logs	Bit	Indica si se loguea el proceso
XSLToutput	Text	XSLT que se le aplica al XML resultado
CreateDate	Datetime	Fecha de creación
ModifiedDate	Datetime	Fecha de última modificación. Se actualiza automáticamente.
DeletedDate	Datetime	Fecha en la que fue borrado

Tabla ProcessInstance

Cada vez que se inicia la ejecución de un proceso se almacena un nuevo registro en esta tabla, en donde se almacenan los datos del mismo. La estructura de esta tabla es la siguiente:

Campo	Tipo	Observación
IDProcessInstance	Char	Identificación de la instancia del proceso
ID	Bigint	Identificador legible de la instancia del proceso
IDProcessDefinition	Char	Identificación de la definición de proceso asociada a la instancia del proceso.
ProcessState	varchar	Estado en el que se encuentra el proceso
MileStone	varchar	Estado del proceso de negocio
StateMessage	varchar	Mensaje del estado
IDSession	char	Identificador de la sesión que creó la instancia.
XMLInput	text	Serialización del formulario de entrada del proceso
ObjectSerialization	text	Serialización del objeto WFProcessInstance asociado a la instancia del proceso.
ExecutionControl	varchar	Indica como continúa la ejecución del proceso
ExecutionControlDate	datetime	Fecha cuando se seteo el ExecutonControl
IDException	varchar	Identificador de Excepción
ExceptionMessage	varchar	Mensaje de Excepción.
OpenedDate	datetime	Se ingresa automáticamente al insertar el registro
ModifiedDate	datetime	Fecha de última modificación. Se actualiza automáticamente se hace un update sobre el registro
ClosedDate	datetime	Se ingresa automáticamente al actualizar el estado, siempre y cuando sea un estado final.

Existe un "trigger" que actualiza la tabla "ProcessInstanceStates" cada vez que se modifique el campo "IDProcessState" o se inserte un nuevo registro. De esta manera se podrá hacer el seguimiento de los cambios de estado que tuvo una actividad.

También existe un trigger que actualiza el campo "ClosedDate" cuando se actualiza un estado que es final, es decir que en la tabla "ProcessStates", el campo "FinalState" vale 1. Adicionalmente, si el registro ya tiene cargado un valor en "ClosedDate", no se permitirá actualizar el valor de IDProcessState.

Tabla ProcessInstanceStates

En esta tabla se almacenan los estados que tienen las instancias de los procesos que corren en el servidor. Dada la identificación de una instancia de proceso se puede obtener cual es su estado. La estructura de esta tabla es la siguiente:

Campo	Tipo	Observaciones
IDProcessInstanceState	Int	Identificación del estado de una instancia de proceso
IDProcessInstance	Char	Identificación de una instancia de proceso.
ProcessState	Varchar	Relacionado con ProcessStates.ProcessState
CreationDate	DateTime	Fecha de creación del registro
StateMessage	Varchar	Mensaje descriptivo del Estado.

Tabla ProcessStates

En esta tabla se almacenan todos los estados que pueden tener las instancias de un proceso.

Campo	Tipo	Observaciones
ProcessState	Varchar	Identificador del estado
FinalState	Bit	Indica si el estado es final. Se utilizará posteriormente para saber si la instancia ha finalizado o no.
Description	varchar	Descripción sobre el estado

Tabla ActivitiesInstance

En esta tabla se almacenan las instancias de las actividades. Cada vez que se crea una nueva instancia de una actividad se almacena en esta tabla.

La estructura de dicha tabla es la siguiente:

Campo	Tipo	Observaciones
IDActivityInstance	Char	Identificación de la instancia de la actividad.
IDProcessInstance	Char	Relacionado con ProcessesInstance.IDProcessInstance
ActivityState	Varchar	Relacionado con ActivityStates.ActivityState
StateMessage	Varchar	Mensaje del estado
Synchronic	Bit	Indica si el proceso es sincrónico o no.
Priority	Int	Prioridad de la actividad
Limit	DateTime	Tiempo limite
IDToolAgent	Varchar	Identificador del ToolAgent que implementa la activity. Relacionado con ToolAgents.IDToolAgent
ObjectSerialization	Text	Serialización del objeto
IDException	Varchar	Identificado de la excepción
ExceptionMessage	Varchar	Mensaje cuando se produce una excepción
OpenedDate	DateTime	Se ingresa automáticamente al insertar el registro
ModifiedDate	DateTime	Fecha de última modificación. Se actualiza automáticamente se hace un update sobre el registro.
ClosedDate	DateTime	Se ingresa automáticamente al actualizar el estado, siempre y cuando sea un estado final.
Title	Varchar	
Description	Varchar	Campo disponible para agregar aclaraciones durante la ejecución de la instancia. Por ejemplo se podría almacenar el motivo de cierre si fuese por error.
ActivityType	Varchar	Tipo de Actividad
UserVisible	Bit	Indica si es visible para el usuario

Tabla ActivityStates

Tabla en donde se almacenan todos los estados que pueden tener las instancias de las actividades.

Campo	Tipo	Observaciones
ActivityState	Varchar	Identificador del estado
Final State	Bit	Indica si el estado es final. Se utilizará posteriormente para saber si la instancia ha finalizado o no.
Description	Varchar	Descripción del estado

Tabla ActivityInstanceStates

En esta tabla se almacenan los estados en los que se encuentran las distintas instancias de las actividades.

Campo	Tipo	Observaciones
IDActivtyInstanceState	Int	Identificador del estado de la instancia de la actividad
IDActivityInstance	Char	Relacionado con ActivityInstance.IDActivityInstance
ActivityState	Varchar	Relacionado con ActivityStates.IDActivityState
CreationDate	DateTime	Fecha de creación del registro
StateMessage	Varchar	Descripción del estado

Tabla para manejo de los ToolAgents

Tabla ToolAgents

En esta tabla se almacenan todos los datos de los tool agents correspondientes al Workflow.

Campo	Tipo	Observaciones
IDToolAgent	Varchar	Identificador del tool agent
Description	Varchar	Descripción del tool agent
Assembly	Varchar	Assembly o Dll en donde se encuentra la clase asociada al tool agent.
Class	Varchar	Clase asociada al tool agent
Help	Varchar	Texto de ayuda sobre la utilización del Tool Agent
WebForm	Varchar	Control web que permite ingresar la info.
Image	Varchar	Imagen a mostrar en el formulario
CreatedDate	Datetime	Fecha de creación
ModifiedDate	Datetime	Fecha de modificación

Tabla para definición de esquemas

Tabla SchemasDefinition

En esta tabla se almacenan las definiciones de esquemas

Campo	Tipo	Observaciones
IDSchemasDefinition	Varchar	Identificador del Esquema
SchemaName	Varchar	Nombre del Esquema
SchemaDefinition	Varchar	XML con la definición del Esquema.
CreatedDate	Datetime	Fecha de creación
ModifiedDate	Datetime	Fecha de última modificación

Tablas para manejo de Participantes

Tabla Participants

En esta tabla se almacenan los datos de los participantes del workflow. Además de la identificación, un nombre y una descripción un participante es un determinado tipo, por esto es que se cuenta con un campo IDParticipantType que asocia a un participante con un tipo de participante.

Campo	Tipo	Observaciones
IDParticipant	Char	Identificador del participante
Name	Varchar	Nombre del participante
Description	Varchar	Descripción sobre el participante
IDParticipantType	Char	Identificador del tipo de participante. Relacionado con ParticipantsType.IDParticipantType
ObjectSerialization	Text	Serialización del objeto
CreatedDate	DateTime	Fecha de creación
ModifiedDate	DateTime	Fecha de modificación

Tabla ParticipantsTypes

En esta tabla se almacenan los tipos de participantes del workflow. Cada participante del workflow es de algún tipo. En particular cada tipo de participante tiene storage asociada, dicho storage es el componente que se utiliza para acceder a los datos. Es por esto que se tiene un campo IDStorageType que asocia a un tipo de participante, con un determinado storage de la tabla StoragesTypes.

Campo	Tipo	Observaciones
IDParticipantType	Char	Identificador del tipo de participante
IDStorageType	Char	Identificador del tipo de storage que usa este tipo de participante. Relacionado con StoragesTypes.IDStorageType
Name	Varchar	Nombre del tipo de participante
Description	Varchar	Descripción del tipo de participante
IDSchemasDefinition	Char	Identificador del schema relacionado con este tipo de participante. Relacionado con la tabla SchemasDefinition.IDSchemaDefinition
ParticipantAssembly	Varchar	Assembly en donde se encuentre la clase que implementa el tipo de participante
ParticipantClass	Varchar	Nombre de la clase dentro del assembly que implementa el tipo participante.
IsContainer	Bit	Sirve para determinar si este tipo de participantes contiene otro participantes
CreatedDate	DateTime	Fecha de creación
ModifiedDate	DateTime	Fecha de modificación

Tabla StorageTypes

Tabla en donde se almacenan los tipos de "Storages" disponibles. Como se vió anteriormente, los storages son componentes mediante los cuales, se puede acceder a la información que se tenga sobre los recursos. Cada Participante pertenece a un determinado tipo dependiendo del componente encargado para acceder a la información sobre los recursos.

Campo	Tipo	Observaciones
IDStorageType	Char	Identificador del tipo de storage
Name	Varchar	Nombre del tipo de storage
StorageAssembly	Varchar	Assembly en donde se encuentra la clase para acceder al storage
StorageClass	Varchar	Clase dentro del assembly que se usa para acceder al storage
InitValues	Text	Valores Iniciales
CreatedDate	DateTime	Fecha de creación
ModifiedDate	DateTime	Fecha de modificación

Tabla ParticipantsRelations

Tabla en donde se almacenan las relaciones entre los distintos participantes del workflow.

Campo	Tipo	Observaciones
IdParticipant	Char	Identificación del participante
IdChildParticipant	Char	Identificación del participante "hijo" del participante
CreatedDate	Char	Fecha de creación

Tabla WorkitemsInstance

Tabla en donde se almacenan las instancias de los ítems de trabajo ("workitems").

Campo	Tipo	Observaciones
IDWorkitemInstanceState	Char	Identificación del ítem de trabajo
IDActivityInstance	Char	Identificador de la instancia de actividad.
IDParticipant	Char	Identificación del participante que realiza el ítem de trabajo
IDSchemaDefinition	Varchar	Esquema de la información a ingresar.
WorkitemState	Varchar	Estado del ítem de trabajo
StateMessage	Varchar	Mensaje de estado
Priority	Int	Prioridad de la tarea
ParticipantName	Varchar	Nombre del participante
NotifyAssignment	Bit	Notificación de asignación
NotifyProgress	Bit	Notificación de progreso
NotifyExpiration	Bit	Notificación de tiempo de expiración
ObjectSerialization	Text	Serialización del objeto
IDException	Varchar	Identificador de Excepción
ExceptionMessage	Varchar	Mensaje de Excepción
Limit	Varchar	Tiempo limite
ExecutionResult	Text	Resultado de la ejecución
ExpirationDate	Datetime	Fecha máxima de Fin
OpenedDate	DateTIme	Fecha de inicio
ModifiedDate	DateTime	Fecha de modificación
ClosedDate	Datetime	Fecha cerrado
Title	Varchar	Titulo
Description	Varchar	Descripción del ítem de trabajo

Tabla WorkitemsInstanceStates

En esta tabla se almacenan los estados de las diferentes instancias de los ítems de trabajo ("workitems")

Campo	Tipo	Observaciones
IDWorkitemInstanceState	Int	Identificador del estado de un ítem de trabajo
IDWorkitemInstance	char	Identificador de una instancia de trabajo. Relacionado con WorkimtesInstance.IDWorkItemInstance
WorkitemState	Varchar	Estado del ítem de trabajo. Relacionado con WorkitemStates.WorkitemState
CreatedDate	DateTime	Fecha de creación
StateMessage	Varchar	Mensaje de estado

Tabla WorkitemStates

En esta tabla se almacenan todos los estados que puede tomar un workitem

Campo	Tipo	Observaciones
WorkitemState	Varchar	Estado que puede tomar un ítem de trabajo
FinalState	Bit	Indica si es un estado final
Description	Varchar	Descripción del estado.

Tablas para páginas web**Tabla WebPages**

En esta tabla es donde se almacenan los datos de las páginas web

Campo	Tipo	Observaciones
IDWebPage	Char	Identificador de la página web
Name	Varchar	Nombre de la página web
Description	Varchar	Descripción de la página web
IDSystem	Char	Relacionada con Systems.IDSystem
IDSecurityObject	Char	Relacionada con SecurityObjects.IDSecurityObjects
IsEnabled	Bit	Indica si está habilitada
HelpLink	Varchar	Link a ayuda
Configuration	Text	Configuración de visualización de la página
CreationDate	Datetime	Fecha de creación
LastAccessDate	Datetime	Ultima fecha de acceso

Tabla WebPagesMenu

Campo	Tipo	Observaciones
IDMenuItem	Char	Identificador del item del menú
IDMenuItemParent	Char	Identificador del item padre
Name	Varchar	Nombre del menú
Description	Varchar	Descripción del menú
ToolTip	Varchar	ToolTip asociado al item del menú
IDSystem	Char	Sistema al que pertenece
IsEnabled	Bit	Indica si está habilitado
IsGroup	Bit	Indica si tienen hijos
IDWebPage	Char	Identificador de la página que abre
CreationDate	Datetime	Fecha de creación
LastAccessDate	Datetime	Fecha de último acceso

Tabla para manejo de sistemas**Tabla Systems**

En esta tabla se almacenan los diferentes sistemas de una empresa. Entendiendo por sistema un conjunto de procesos de negocios

Campo	Tipo	Observaciones
IDSystem	Char	Identificador del sistema
SystemName	Varchar	Nombre del sistema
SystemAssembly	Varchar	Assembly donde se encuentra la clase que valida sesiones para el sistema
SessionClass	Varchar	Clase encargada de validar sesiones
IsEnabled	Bit	Indica si está habilitado el sistema
CreatedDate	Datetime	Fecha de creación
ModifiedDate	Datetime	Fecha de última modificación

Tabla SystemParameters

En esta tabla se almacenan los parámetros de los diferentes sistemas.

Campo	Tipo	Observaciones
IDSystem	Char	Identificador del sistema
ParamKey	Varchar	Nombre del parámetro
ParamValue	Varchar	Valor del parámetro
CreatedDate	Datetime	Fecha de creación
ModifiedDate	Datetime	Fecha de última modificación

Tabla SystemSessions.

Campo	Tipo	Observaciones
IDSession	Char	Identificador de sesión
IDSystem	Char	Identificador del sistema al que pertenece la sesión
Profile	Text	Perfil del usuario
CreationDate	Datetime	Fecha de creación
LastAccessDate	DateTime	Fecha de último acceso

10 REFERENCIAS DE INTERÉS:

- David Hollingsworth - TC-1003 Workflow Reference Model – WfMC – 1995
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- Orfali, D. Harkey, J. Edwards - Cliente/Servidor Guia de supervivencia – Segunda Edición
Editorial Mc Graw Hill - 1998
- WfMC – TC-1011 Terminology & Glosary - 1999
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- David Hollingsworth WfMC – TC-1023 Common Object Model - WfMC - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC Work Group 1 – TC-1016-X Process Definition Interchange - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC Work Group 1 – TC-1016-P Process Definition Interchange ,Process Model - 1999
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC Work Group 1 – TC- 1016-O Process Definition Interchange – Organizational Model - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC-1025 Process Definition Interface – XML Process Definition Language - 2002
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC – 1009 Aplication Programming Interface (WAPI) – Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC- 1013 Aplication Programming Interface – Naming Conventions - 1997
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC – 1012 Interoperability – Abstract Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – TC -1015 Audit Data Specification - 1998
<http://www.wfmc.org/standards/docs.htm> - 20/5/2004
- WfMC – Workflow Handbook 2003 - 2003
<http://www.wfmc.org/information/handbook03.htm> - 20/5/2004
- WfMC – Workflow Handbook 2002 - 2002
<http://www.wfmc.org/information/handbook02.htm> - 20/5/2004
- WfMC – Workflow Handbook 2001 - 2001
<http://www.wfmc.org/information/handbook01.htm> - 20/5/2004
- www.XData.com (XData)
- www.wfmc.org (Workflow Managament Coalition) - 20/5/2004
- www.urudata.com (QFlow) - 20/5/2004

- Tom Archer, Andrew Whitechapel - Inside C# - Editorial Microsoft Press - 2002