Capítulo 4. Implementación de ACQUA

Capítulo 4

Implementación de ACQUA

Este capítulo describe los detalles de implementación del sistema ACQUA. El objetivo de la configuración del sistema es validar nuestra propuesta y particularmente las propiedades de adaptabilidad que ofrece el sistema.

Se escogió el servicio de mensajería de JAVA (JMS) versión 1.3.1 como una infraestructura de base para la construcción de ACQUA. Este servicio provee una plataforma síncrona y asíncrona de comunicación que permite cubrir las necesidades de comunicación durante la etapa de ejecución de workflows y de modificación de políticas respectivamente. Se utilizó el lenguaje de programación JAVA versión 1.4.1_01. Este lenguaje provee librerías de funciones básicas como el manejo de vectores, construcción de interfaces gráficas, manejo de bases de datos que facilitaron la implementación de ACQUA.

Para el soporte de persistencia se utilizó un SGBD orientado a objetos (FastObjects e7 *Trial Edition*) que permitió el almacenamiento, actualización y recuperación de objetos de una manera clara y sencilla. El motivo de la elección es que este producto ofrece soporte de persistencia para cualquier aplicación desarrollada en JAVA, lenguaje elegido para la implementación.

El resto del capítulo está organizado de la siguiente manera. La Sección 4.1 presenta la arquitectura del motor de ejecución de ACQUA, describe las funciones principales del sistema ACQUA: inicialización, ejecución y adaptabilidad. La Sección 4.2 describe la implementación de las funciones del sistema ACQUA. La Sección 4.3 corresponde a la validación experimental que se realizó para probar la funcionalidad del sistema. Finalmente en la Sección 4.4 se concluye el capítulo y se discuten los aspectos relevantes de la implementación.

4.1 Arquitectura del motor de ejecución

La Figura 4.1 presenta la arquitectura del motor de ejecución de ACQUA conformada de tres componentes: coordinador, administrador de listas (*Worklist Manager*) y administrador de agentes (*Agent Manager*).

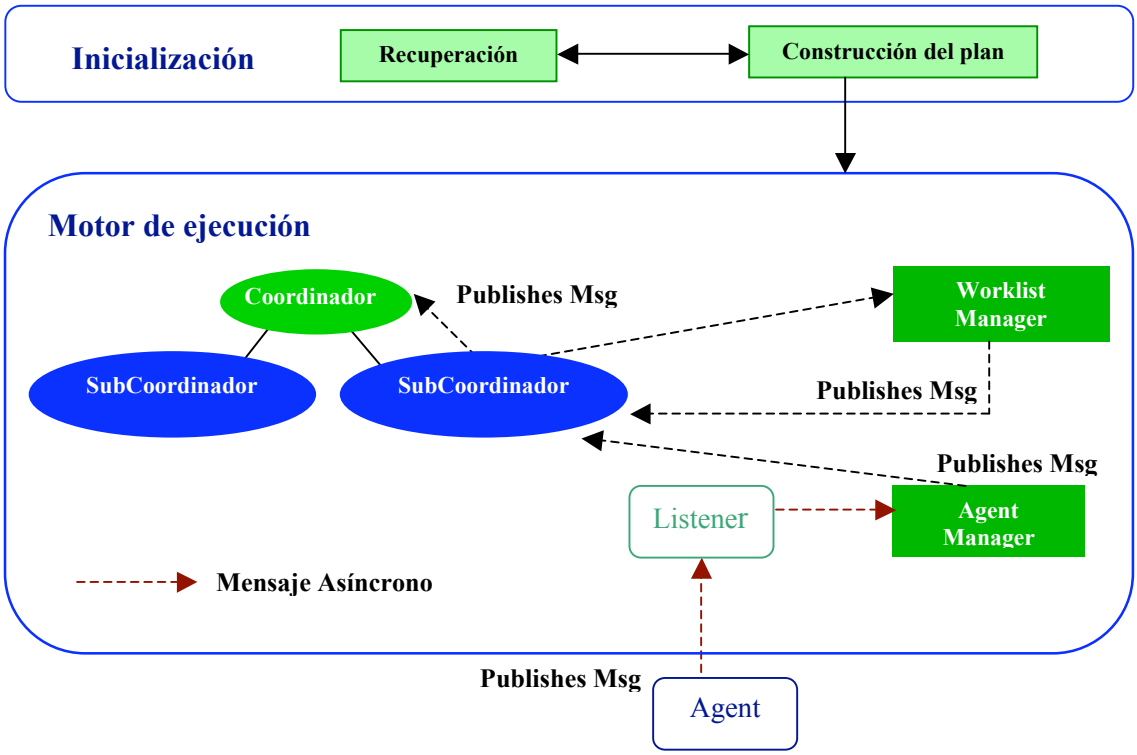

Figura 4.1: Arquitectura del motor de ejecución de ACQUA

Cada **coordinador** (cada nodo operador del grafo) se suscribe a un `topic`, que es un objeto del JMS que un cliente utiliza para especificar el origen de los mensajes que produce y el destino de los mensajes que consume [27]. El **coordinador** (la raíz del grafo del plan de ejecución de workflows), los **subcoordinadores** (operadores del workflow), las listas de trabajo (***Worklist Manager***) y el administrador de agentes (***Agent Manager***) deben suscribirse a un `topic` cada uno. La Figura 4.2 muestra la relación existente entre clientes y `topics` para enviar y recibir mensajes.

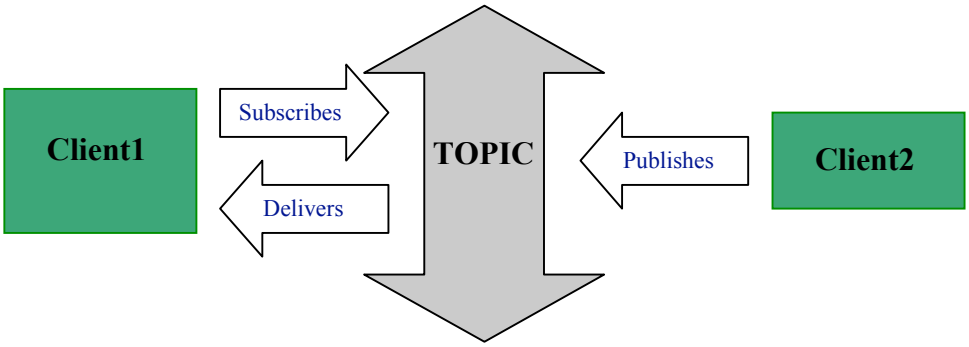

Figura 4.2: Envío y recepción de mensajes

En tiempo de ejecución, el coordinador publica mensajes que son recibidos por los administradores de listas de trabajo para notificarles la acción a realizar dado el resultado de la evaluación de las condiciones. Los administradores de listas de trabajo publican mensajes al `topic` del coordinador para notificarle el estado de la ejecución de la actividad, así mismo el administrador de agentes notifica al coordinador la disponibilidad de agentes para que ejecuten las actividades. El administrador de agentes recibe mensajes a través de `listeners` que le notifican la conexión y desconexión de agentes.

4.1.1 Inicialización

La fase de inicialización es la etapa previa a la ejecución. La Figura 4.3 muestra las actividades del proceso de inicialización.

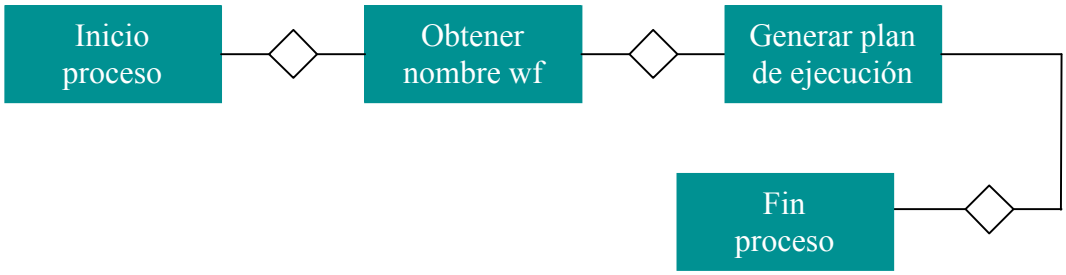

Figura 4.3: Proceso de inicialización

La construcción del plan de ejecución inicia con la recuperación de los objetos almacenados en la base de datos de la herramienta de definición TERRA [26], se obtienen las actividades, las apolíticas y el flujo, es decir la secuencia en la que se ejecutarán las actividades. Finalmente se crean los objetos del grafo de ejecución.

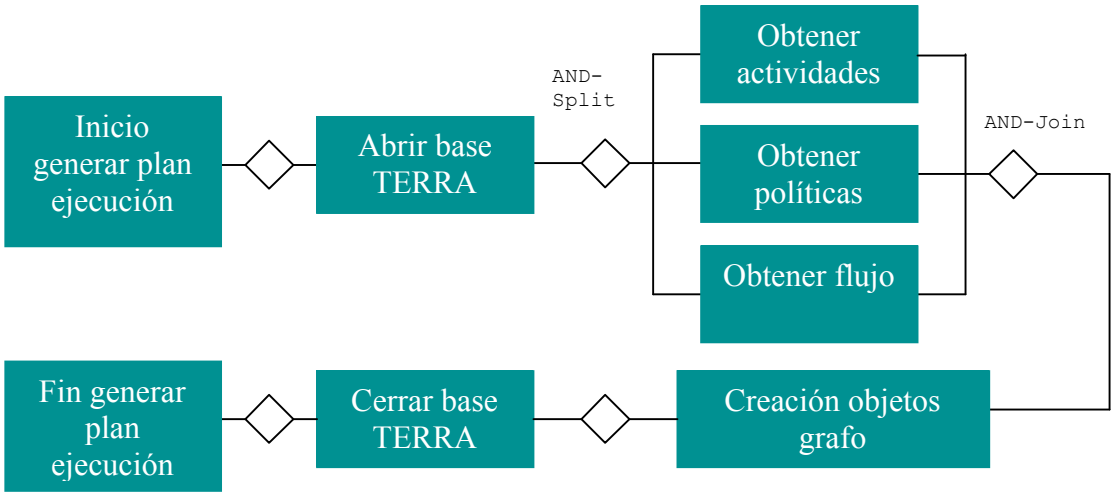

Figura 4.4: Proceso de generación del plan de ejecución

4.1.2 Ejecución

Para la fase de ejecución se tienen dos opciones si el plan se generó en ejecuciones anteriores o si es la primera vez que se ejecutará el workflow. Primero se obtiene el nombre del plan de ejecución para recuperar el plan de ejecución correspondiente de la base de ACQUA. Una vez recuperado se ejecuta.

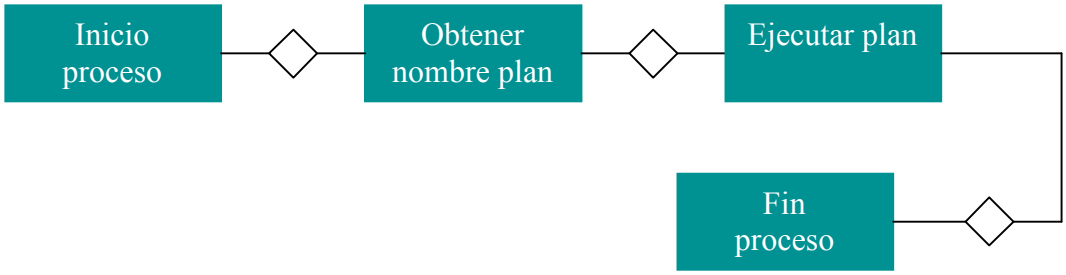

Figura 4.5: Recuperación y ejecución de un plan

La siguiente Figura muestra el workflow para la ejecución de un plan por primera vez, es decir se debe generar primero el plan para posteriormente ejecutarse.

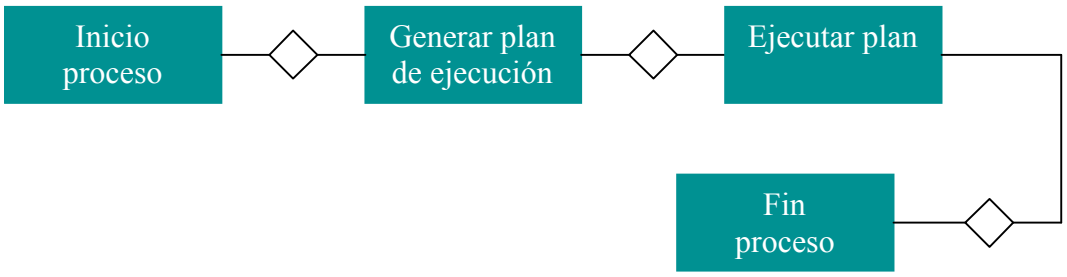

Figura 4.6: Generación y ejecución de un plan

La ejecución de un plan de ejecución inicia con la creación de los administradores de listas y los administradores de agentes. Primero se obtiene el primer operador, se recupera la actividad y se manda al administrador de agentes que a su vez manda la actividad al administrador de listas para que finalmente se ejecute. Este proceso se repite el número de operadores existentes mientras no haya algún problema durante la ejecución que la detenga.

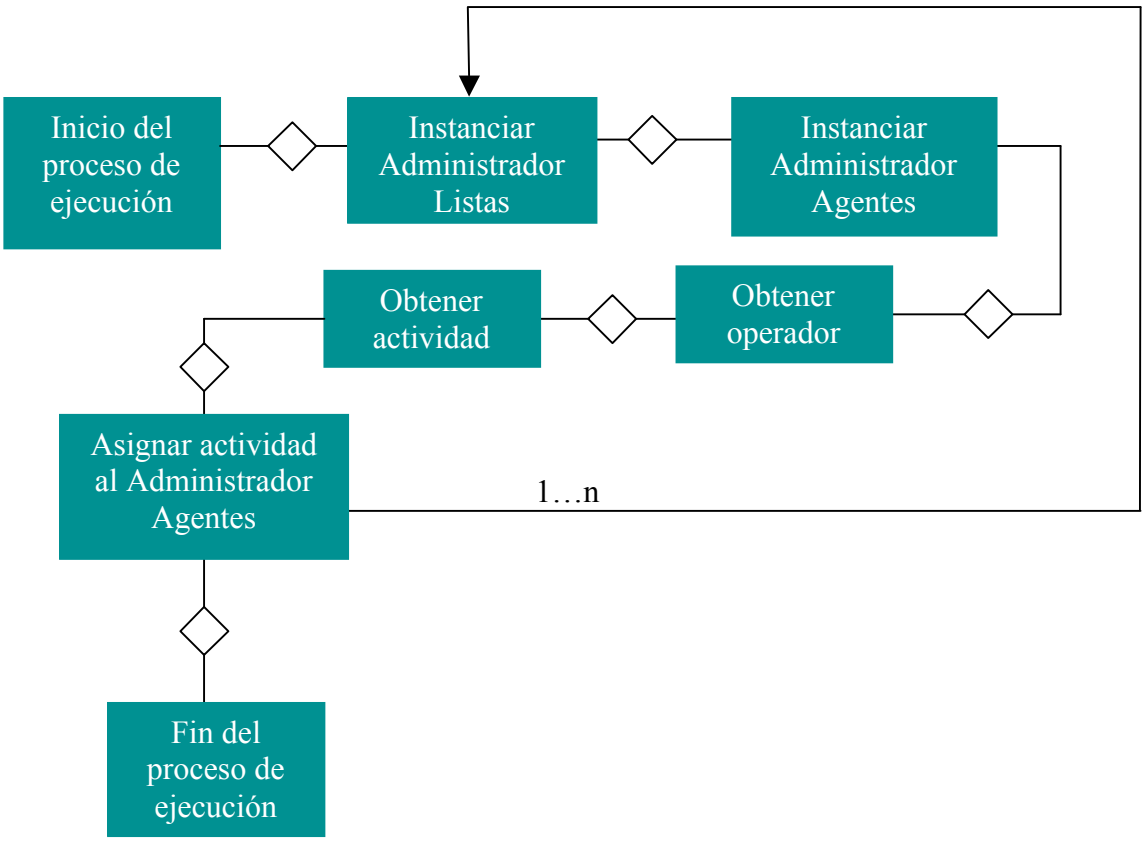

Figura 4.7: Proceso de ejecución de un plan

4.1.3 Adaptabilidad

Finalmente ACQUA ofrece al usuario la opción de poder modificar las políticas de ejecución del workflow en tiempo de ejecución. ACQUA manda el plan de ejecución a modificar a través del JMS y la interfaz de adaptabilidad se encarga de ejecutar el proceso de modificación. El siguiente workflow muestra la serie de actividades que ACQUA realiza para modificar un plan de ejecución.

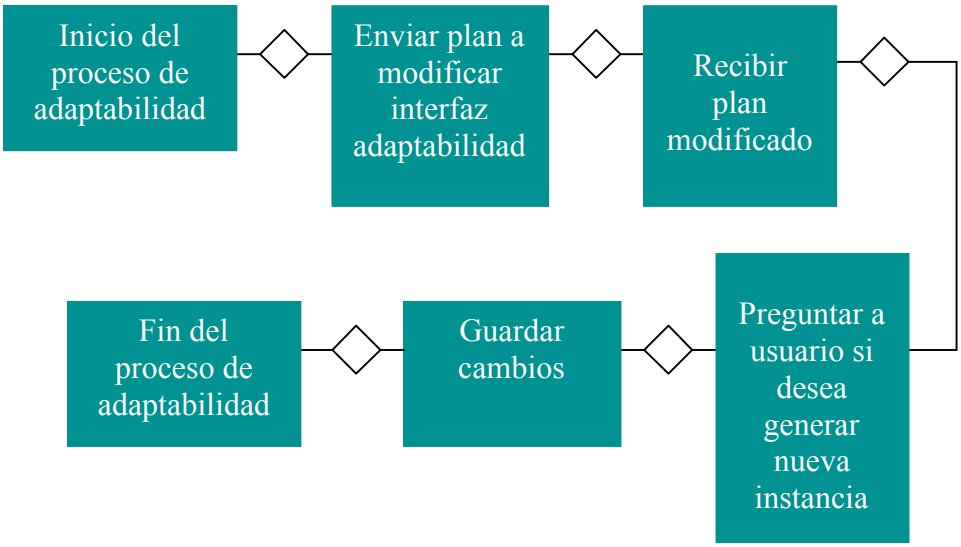

Figura 4.8: Adaptabilidad ACQUA

La interfaz de modificación ACQUAdaptabilidad al recibir el plan a modificar debe realizar una serie de actividades para finalmente regresar el plan modificado. El siguiente workflow presenta el proceso de adaptabilidad de ACQUAdaptabilidad.

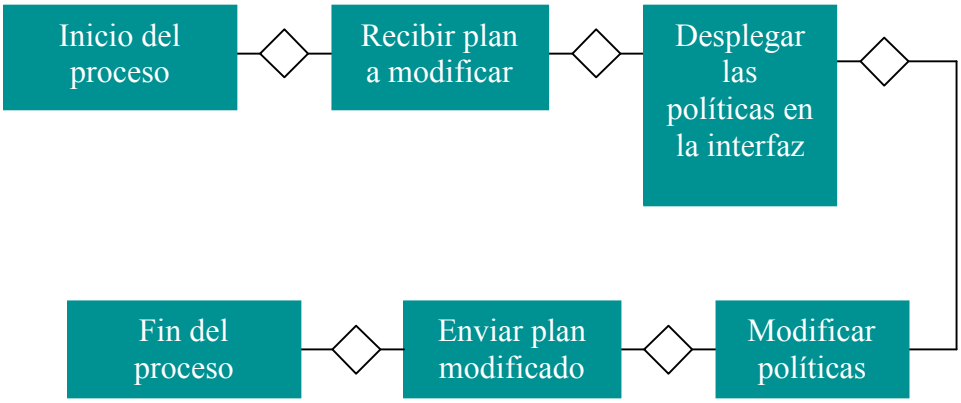

Figura 4.9: Adaptabilidad ACQUAdaptabilidad

4.2 Implementación ACQUA

4.2.1 Inicialización

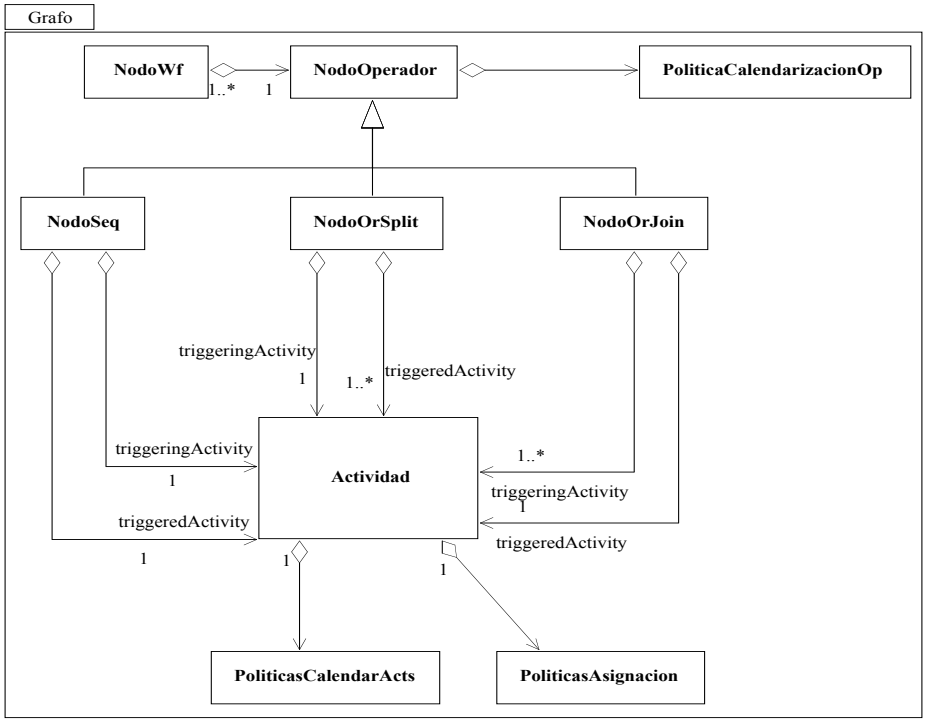

Figura 4.10: Paquete Grafo

Las clases programadas para llevar a cabo el proceso de inicialización son las siguientes:

`Grafo`*:* Implementa la construcción del plan de ejecución para lo cual establece la conexión con la base de datos de TERRA [26]. Además de la construcción también implementa un algoritmo para dibujar el grafo con el que se representa el workflow a ejecutar.

`NodoWf`: Implementa el grafo (plan de ejecución del workflow). Está formado por operadores que a su vez están formados por actividades.

`NodoOperador`: Superclase abstracta de los operadores de orden.

`PoliticaCalendarizacionOp`: Corresponde a las políticas propias de los operadores de orden como el tiempo de activación durante la ejecución.

`Actividad`: Contiene atributos de las actividades que conforman un workflow.

`PoliticasCalendarActs`: Contiene los atributos correspondientes al proceso de calendarización de actividades tales como: *triggeringInstant*, *preconditionEvaluationInstant*, *postconditionEvaluationInstant*, etc. También las políticas referentes al tiempo de permanencia en las listas de trabajo.

`PoliticasAsignacion`: Sus atributos son las políticas correspondientes a la asignación de una actividad a un agente para que la ejecute.

Los métodos involucrados en la construcción del plan de ejecución son los siguientes:

`comenzarConstruccion`: Método que invoca el método de recuperar el flujo, los agentes definidos de la base de TERRA [26] y finalmente llama al método de construirGrafo.

`construirGrafo`: Inicia la construcción de grafo, este método recibe la raíz del grafo y el flujo recuperado de TERRA [26]. Con base en el flujo se crean los objetos de tipo `Actividad`, los objetos de tipo operador y los objetos correspondientes a las políticas de ejecución.

`dibujaGrafo`: Dibuja el grafo mediante un algoritmo que recorre el grafo y dibuja cada uno de los nodos operadores con sus respectivas actividades y subworkflows.

El siguiente algoritmo fue utilizado para la construcción del grafo de acuerdo a la definición de un workflow.

```
Inicio
Wf<-raíz del grafo

//Se comienza a construir el primer nivel del grafo
//Las actividades que lanzan (triggering) y las actividades lanzadas
(triggered) se almacenan en un vector ordenado que es utilizado para
crear los operadores de orden
```

```
Actividad [numeroActividades] triggeringActivity[];

//El workflow comienza con una actividad
int numeroTriggering=1;
triggeringActivity[0]<-obtenerPrimeraActividad();
do{
      obtener el siguiente operador OP
      //Crear el objeto de tipo operador dependiendo del tipo
      //Se liga el operador al vector de operadores de la raíz
      operadores.add(OP);

      //Se guardan las triggering/triggered y subworkflows dependiendo
      del tipo de operador

      for(int j=0;j<numeroTriggering;j++){
            if(OP==SEQ){
                  OP.triggeringActivity <- triggeringActivity[j];
                  OP.triggeredActivity <- obtenerSiguienteAct();
                  numeroTriggering++;
                  triggeringActivity[0] <- OP.triggeredActivity;
            }
            else
            if (OP==OR-SPLIT){
                  OP.triggeringActivity <- triggeringActivity[0]
                  int numeroRamas <- obtenerNumeroRamas();
                  int k=0;
```

```
                    //Por cada rama del operador split
                    {     //Si la rama contiene más de 1 actividad
                          //se debe crear un subWorkflow y
                          //ligarlo al nodo principal, por lo que se
                          //debe llamar recursivamente este //algoritmo
                          if(numeroRamas>1){
                                subWf <- crearSubWF();
                                OP.triggeredActivity[k++] <- subWf;
                                triggeringActivity[k++] <- subWf;
                          }
                          else{
                                OP.triggeredActivity[k++]<-
                                obtenerActividadRama();
                                triggeringActivity[j++]<-
                                obtenerActividadRama();
                          }
                    }numeroTriggering=k-1;
              }//del operador Split
              else
                if (OP==OR-JOIN){
                    for(int j=0;j<numeroTriggering;j++){
                          OP.triggeringActivity <- triggeringActivity[j];
                    }
                    OP.triggeredActivity <- obtenerSiguienteAct();
                    numeroTriggering=1;
                    triggeringActivity[0]<- obtenerSiguienteAct();
                }
        }//del for
}while(no sea el final del workflow)
```

4.2.2 Ejecución

Las clases programadas para llevar a cabo el proceso de ejecución pertenecen a dos paquetes: `Grafo` y `Motor`. Las clases programadas del paquete `Motor` son las siguientes:

`AdministradorAgentes`: Recibe cada una de las actividades del workflow y con base en las políticas de asignación busca el agente idóneo que lleve a cabo la ejecución. Una vez encontrado un agente envía el nombre del agente seleccionado y la actividad al administrador de listas.

`AdministradorListas`: Agrega las actividades a las listas de los agentes y lanza a ejecutar cada una de las actividades de la lista correspondiente utilizando la política FIFO.

`Agente`: Tiene las características del agente definidas en la herramienta TERRA [26] tales como: nombre del agente, puesto y roles.

`Lista`: Son las listas de actividades asignadas a los agentes y manejadas por el administrador de listas. Tiene como atributos el nombre del agente y un `Vector` de actividades.

Existe una relación de dependencia entre los objetos del paquete `Grafo` y el paquete Motor, esto es la raíz del grafo (**`NodoWf`**) es la única que puede crear instancias de la clase **`AdministradorAgentes`** y **`AdministradorListas`** pertenecientes al paquete `Motor`.

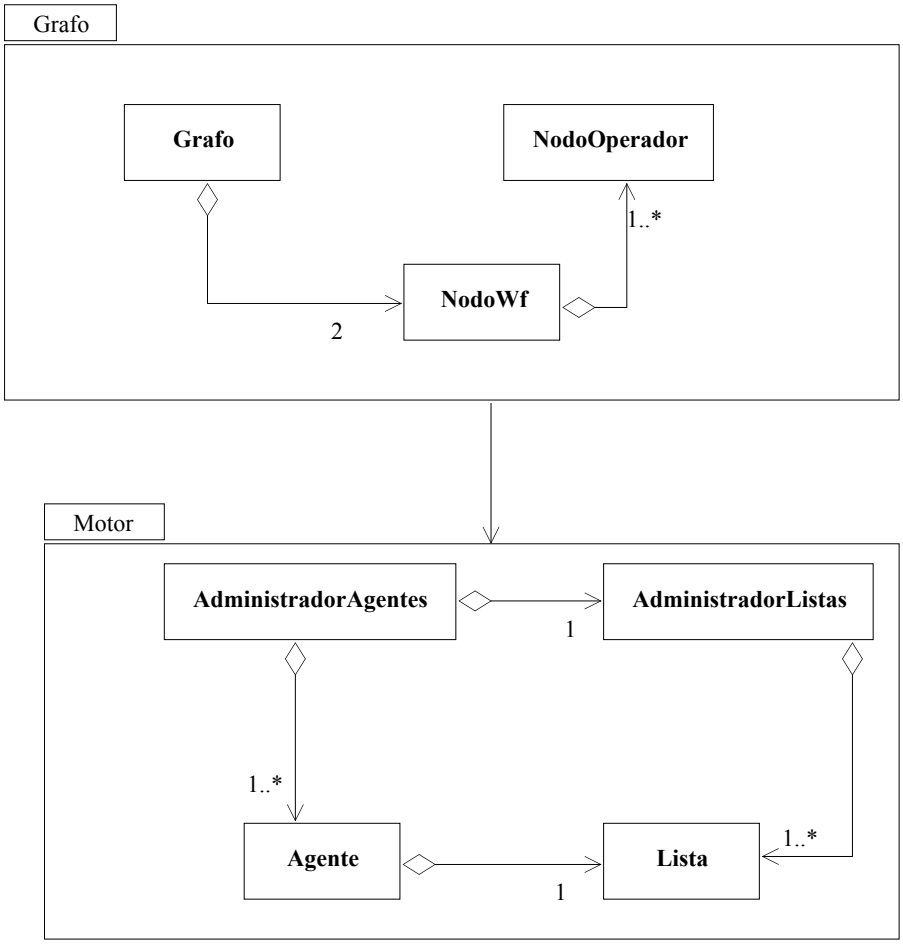

Figura 4.11: Paquete Grafo y Motor

De acuerdo a la arquitectura del motor de ejecución, en el proceso de ejecución intervienen un coordinador y varios subcoordinadores dependiendo del número de operadores del workflow. Los tipos de operadores implementados en ACQUA son: `SEQ`, `OR-SPLIT` y `OR-JOIN`. Cada uno de estos funge como subcoordinador durante la ejecución dirigidos por un coordinador general, la raíz del grafo, por lo que los métodos involucrados en la ejecución fueron implementados en cada uno de estos operadores.

`triggeringActivityHandler`: Manda a evaluar las precondiciones de cada una de las actividades (*triggeringActivities*), si no hay error invoca al método `lanzarActividad`.

`evaluaPrecondicion`: Simula la evaluación de la precondición de una actividad.

`precondicionFailure`: Ejecuta las operaciones necesarias si la precondición falló basándose en las políticas correspondientes.

`lanzarActividad`: Método que lanza una actividad de acuerdo a las políticas de comportamiento para ello manda la actividad al administrador de agentes, espera el resultado de la ejecución e invoca al método `evaluaPostcondicion` y `postcondicionFailure` en caso de error.

`evaluaPostcondicion`: Simula la evaluación de la precondición de una actividad.

`postcondicionFailure`: Ejecuta las operaciones necesarias si la poscondición falló basándose en las políticas correspondientes.

`esperoMensaje`: Crea el objeto de tipo `Topic` donde el operador recibe mensajes. Cabe mencionar que los mensajes provienen del administrador de listas únicamente.

`mandarMensaje`: Crea el objeto de tipo `Topic` destino. Recibe parámetros que le permiten mandar diversos tipos de mensajes a las listas de trabajo dependiendo de las políticas de *precondition failure* y *postcondition failure*

`mandarMensajeNodoWf`: Crea el objeto de tipo `Topic` destino. Recibe parámetros que le permiten mandar diversos tipos de mensajes a la raíz del grafo dependiendo de la política de *matching* establecida.

`triggeredActivityHandler`: Manda a evaluar las precondiciones de cada una de las actividades (*triggeredActivities*), si no hay error invoca al método `lanzarActividad`.

El nombre de los métodos de los operadores es el mismo, sin embargo su operación es diferente dependiendo del tipo de operador porque no todos los operadores tienen el mismo número de *triggeringactivities* y *triggeredactivities* .

Para el caso de las clases del paquete `Motor` los métodos fundamentales de la ejecución son:

De la clase **`AdministradorAgentes:`**

`asignacionActividad`: Método de la clase `AdministradorAgentes` que recibe la actividad que se manda al administrador de listas.

De la clase **`AdministradorListas`** los métodos más importantes son los siguientes:

`agregarActividad:` Método de la clase `AdministradorListas` que recibe la actividad y el nombre del agente. Agrega la actividad a la lista correspondiente.

`asignarActividadAgente`: Método que selecciona las actividades de la lista y procesa su ejecución.

`mandarMensaje`: El administrador de listas manda mensajes al operador dependiendo del estatus de la ejecución de una actividad.

`esperoMensaje`: Crea el objeto de tipo `Topic` donde el administrador de listas recibe mensajes.

`realizoAccion`: Dependiendo del mensaje recibido por el coordinador (*precondition failure* y/o *postcondition failure*) ejecuta una operación acorde con las políticas de ejecución.

4.2.3 Adaptabilidad

Para implementar la adaptabilidad fue necesario desarrollar una interfaz independiente a la ejecución que permitiera al usuario cambiar las políticas de manera sencilla. Los métodos de la clase `ACQUAdaptbilidad` permiten recibir el plan de ejecución, desplegar las políticas en la interfaz, modificar las políticas y finalmente regresar el plan modificado. Los métodos de `ACQUAdaptabilidad` son los siguientes:

`esperoMensaje`: Crea el objeto de tipo `Topic` para poder recibir mensajes de ACQUA.

`onMessage`: ACQUAdaptabilidad es implementa la interfaz `MessageListener` por lo que tiene este método que detecta los mensajes asíncronos enviados por ACQUA.

`buscarActividad`: Recibe la actividad seleccionada para ser modificada y realiza una búsqueda a través del plan de ejecución.

`modificarActividad`: Recibe la actividad modificada y actualiza la actividad en el plan de ejecución.

`mandarMensajeAcqua`: Manda el objeto modificado de regreso a la interfaz ACQUA.

4.3 Validación experimental

El objetivo principal de esta validación experimental es probar las funcionalidades de ACQUA mediante la ejecución de una aplicación de comercio electrónico representativa con la finalidad de mostrar la ejecución de este proceso de negocio.

4.3.1 El sistema CD e-Store

Se especificó la aplicación CD e-Store, una tienda virtual de discos compactos destinada a usuarios pertenecientes a una comunidad. La aplicación consta de las siguientes funciones:

- Ingreso de usuarios registrados al sistemas mediante la validación de un *login* y *password*. Los usuarios pertenecientes a una comunidad cuentan con un nombre de identificación y una clave secreta que les permite ingresar a la tienda. Si el usuario no se encuentra registrado puede ingresar sus datos al sistema y automáticamente se le asigna un número y una clave.

- Acceso a múltiples catálogos de productos. CD e-Store cuenta con catálogos de diversos tipos de música, el usuario escoge el tipo de música de su interés y el sistema despliega el catálogo correspondiente.

- Compra electrónica mediante el ingreso de una cuenta bancaria. Para adquirir un producto, se le pide al usuario ingrese un número de cuenta bancaria para poder hacer el cargo correspondiente.

- Mensajes de confirmación y error al usuario. El sistema constantemente notifica al usuario el estado de sus acciones.

- Envío de productos adquiridos. Una vez adquirido el producto, CD e-Store se compromete a entregar el producto.

4.3.2 Ejecución de una compra electrónica

La Figura 4.12 muestra el workflow de la compra realizada a través de la tienda virtual de discos compactos:

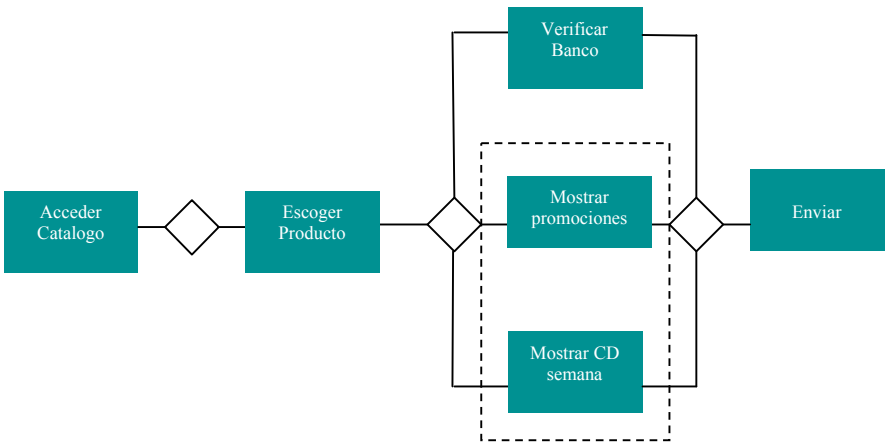

Figura 4.12: Compra electrónica mediante CD e-Store

La actividad **AccederCatalogo** realiza la tarea de verificar que los datos (*login* y *password*) pertenezcan a un usuario registrado en el sistema. Escoger Producto muestra el catálogo de productos y permite que el usuario escoja el producto que desea comprar.

La actividad **VerificarBanco** se encarga de realizar los trámites bancarios, verificar el número de cuenta y realizar el cargo correspondiente.

MostrarPromociones informa al usuario de las promociones vigentes en la tienda.

MostrarCDSemana informa al usuario cuál fue el disco compacto más vendido durante la semana.

Enviar procesa el envío de productos al cliente.

El plan de ejecución del workflow de compra quedó de la siguiente forma:

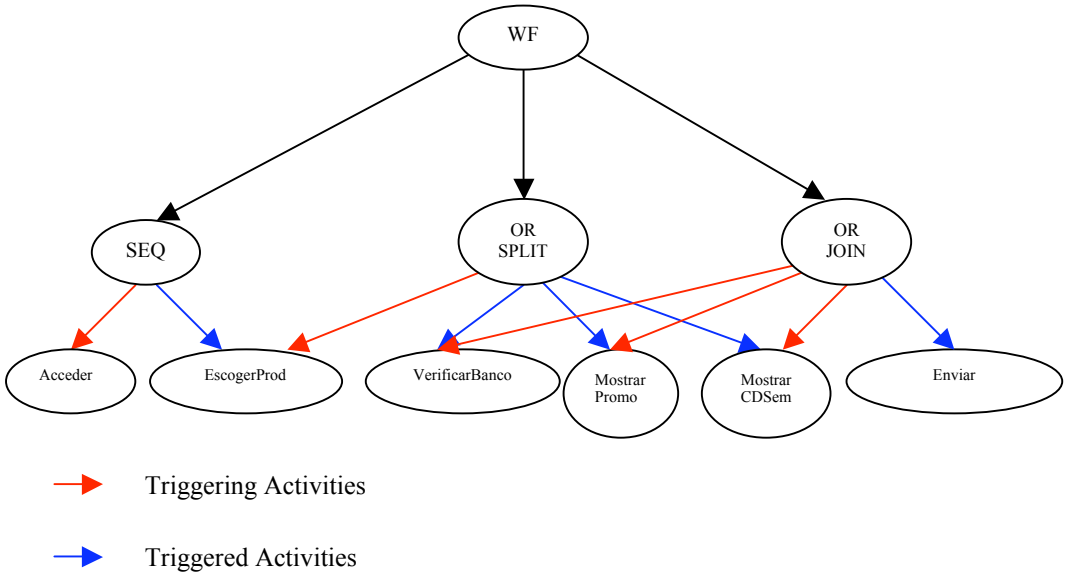

Figura 4.13: Grafo de compra electrónica CD e-Store

ACQUA comienza el proceso de ejecución lanzando cada una de las actividades del Workflow. Para el caso de la actividad **AccederCatalogo** el agente seleccionado sería el método **verificadorUsuarios**, una vez elegido el agente y si los datos fueron correctos el se despliega la ventana de escoger productos del catálogo (EscogerProductos) y así sucesivamente hasta llegar al final del workflow mientras no exista algún problema durante la ejecución de alguna actividad y las políticas lo permitan.

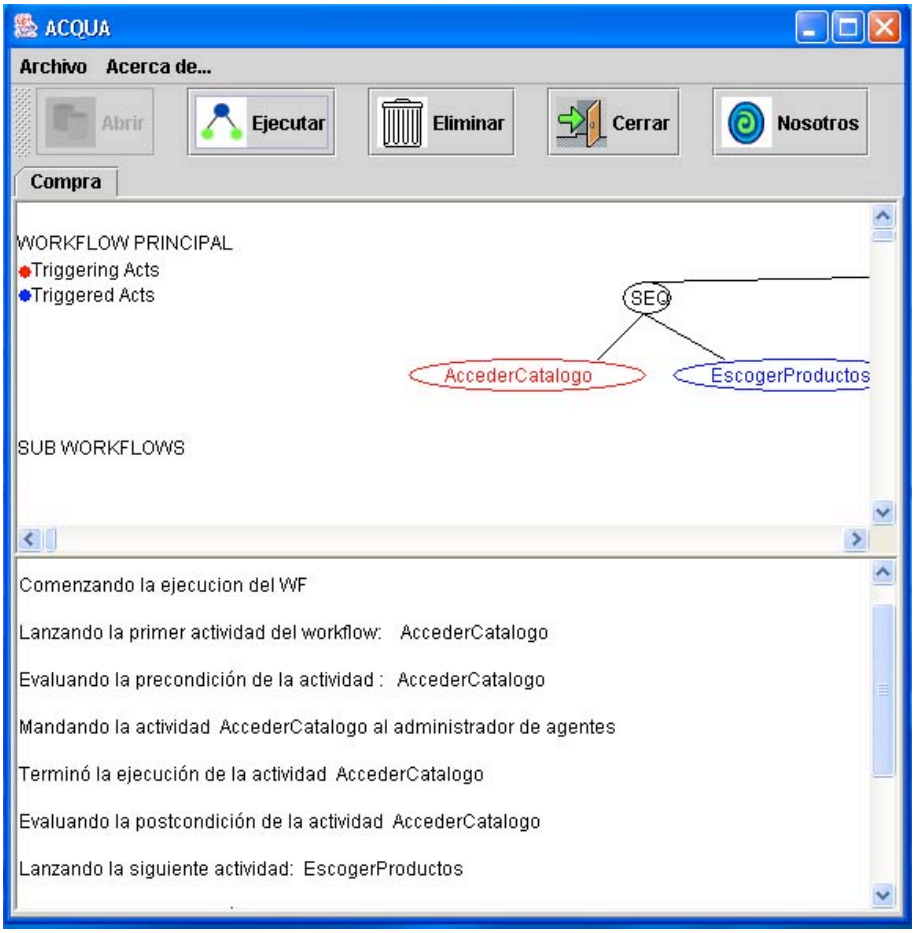

Figura 4.14: Ejecución de Actividades

Una vez que el usuario ingresó al sistema, escoge un producto y se despliega la ventana correspondiente que le permite escoger un producto.

Figura 4.15: Catálogo de productos de CD e-Store

La ventana de CD de la semana se despliega solamente una vez durante la ejecución. La siguiente Figura ilustra la ventana del CD de la semana.

Figura 4.16: Promociones de productos de CD e-Store

4.3.3 Modificación de la compra electrónica

Si se desean modificar las políticas de **AccederCatalogo**, ACQUA despliega la interfaz de modificación con las políticas actuales en color verde, de esta forma el usuario puede ver las políticas y decidir el cambio que realizará. Para el caso de la compra electrónica se modificaron tres actividades en cuanto al instante de disparo y al criterio de selección de agentes. La siguiente Figura ilustra la interfaz de adaptabilidad, las políticas que se muestran son las correspondientes a AccederCatalogo.

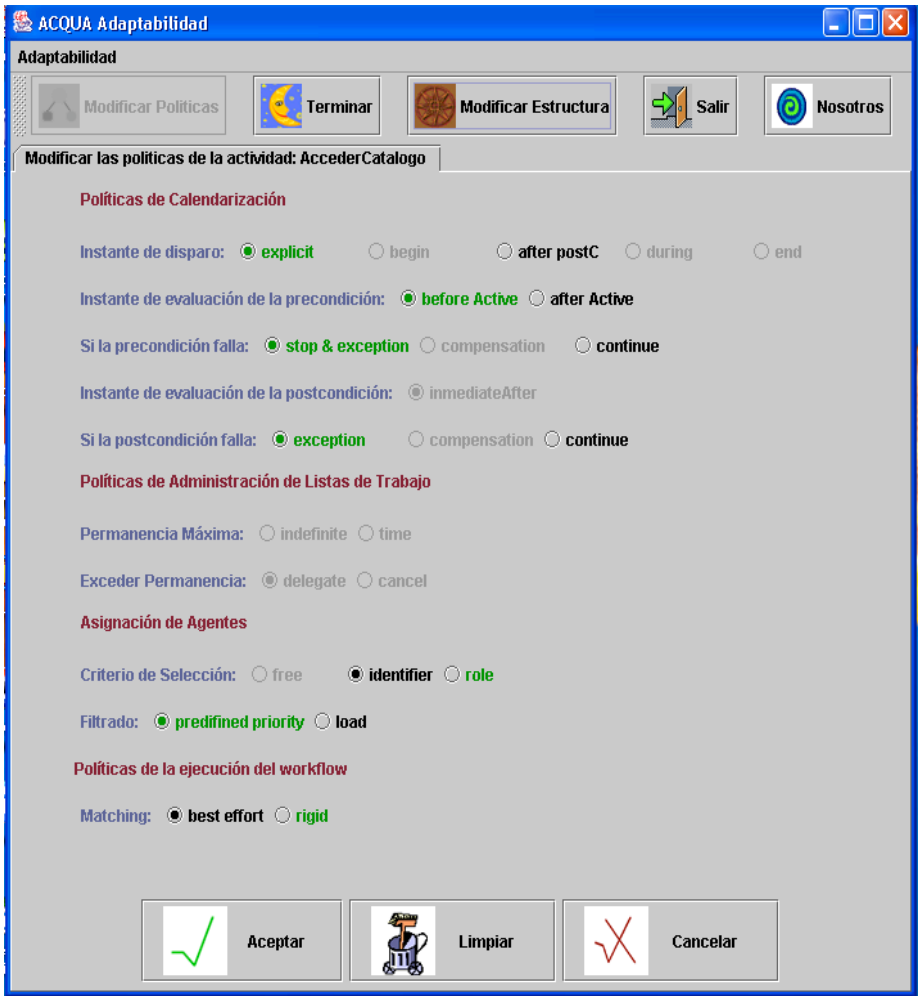

Figura 4.17: Modificación de políticas

4.4 Conclusión

La implementación de una primera versión del prototipo ACQUA muestra la materialización de un sistema de ejecución de workflows que cuenta con mecanismos de ejecución y cuya principal aportación es agregar propiedades de adaptabilidad dinámica mediante la instrumentación de una interfaz independiente al proceso de ejecución.

La validación teórico-práctica de nuestra propuesta se realizó a través de la simulación de la ejecución de un proceso de compra electrónica que permite evaluar nuestro prototipo.