CUMBIA XLM: UN MODELO EXTENSIBLE PARA LA DEFINICIÓN DE LENGUAJES GRÁFICOS ASOCIADOS CON PROCESOS

DAVID MURILLO MATALLANA

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
2006

CUMBIA XLM: UN MODELO EXTENSIBLE PARA LA DEFINICIÓN DE LENGUAJES GRÁFICOS ASOCIADOS CON PROCESOS

DAVID MURILLO MATALLANA

Trabajo de Grado presentado como requisito para optar al título de Magíster en Ingeniería de Sistemas y Computación

Director: Jorge Villalobos Salcedo
Profesor Asociado

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
2006

Tabla de Contenido

Tabla de Figuras

1. Introducción

Este capítulo expone el problema que se desea abordar, los objetivos generales y específicos, la organización del resto del documento y un contexto donde se da una visión global de la intención del proyecto Cumbia.

1.1. El Problema Objeto de Estudio

Es claro que las organizaciones cambian o evolucionan con el tiempo. Esto implica que el software que utilizan estas organizaciones debe evolucionar de forma similar. Sin embargo los cambios que ocurren en las organizaciones o en los problemas que abordan las herramientas de software causan que estas se vuelvan muy complejas hasta el punto de ser poco mantenibles.

Este problema ha creado la necesidad de buscar tecnologías que permitan desarrollar arquitecturas de software flexibles. Esta flexibilidad se refiere a la posibilidad de tomar decisiones sobre la arquitectura lo más tarde posible en el ciclo de vida del software.

De esta búsqueda han surgido conceptos como servicios, composición de servicios, arquitecturas de integración de aplicaciones, orquestación, coreografía, sistemas manejadores de workflows (WFMS – Workflow Management Systems), modelación de procesos de negocio (BPM – Business Process Modeling), entre otros.

El enfoque de WFMS y BPM consiste en definir los procesos y actividades que se realizan en una organización permitiendo utilizar diferentes aplicaciones que lleven a cabo dichas actividades o procesos. Este enfoque permite la comunicación y coordinación entre las personas y tecnologías para llevar a cabo un proceso que puede ser dividido en actividades [BT98].

Definir o diseñar un proceso consiste en especificar las actividades, el orden de ejecución de ellas, los datos que entran y salen de cada actividad y los recursos que necesitan dichas actividades. El diseño de procesos es independiente de las tecnologías o sistemas externos que se pueden utilizar para llevar a cabo las actividades de un proceso. Esto hace que se pueda cambiar un sistema externo por otro y el proceso global no cambie. También, es relativamente fácil agregar nuevas actividades a un proceso existente.

La ventaja de este enfoque es que separa la lógica de los procesos de su implementación, dando lugar a arquitecturas flexibles.

Sin embargo estas aproximaciones tienen sus limitaciones. La mayoría de los WFMS son poco extensibles porque los mecanismos que proveen para tal fin no son suficientes para modelar procesos de diferentes dominios. Según estudios como [Aal03], [ABHK00], [ADHW03], [AHKB00], y [WADH03], no tienen la capacidad de expresión suficiente para dar apoyo a la mayoría de los patrones de workflow [AHKB00] y tienen dependencias tecnológicas que restringen la comunicación con sistemas externos.

Además, estos sistemas difieren en los conceptos y lenguajes utilizados, y en el nivel de aplicabilidad que proveen. Es decir, los WFMS están orientados a dominios específicos.

El proyecto Cumbia (http://agamenon.uniandes.edu.co/~cumbia) inició a mediados del año 2004 en la Universidad de los Andes. El objetivo de este proyecto es proveer una plataforma tecnológica, basada en modelos formales, sobre la cual se pueden construir sistemas manejadores de procesos automatizados aplicables a diferentes dominios.

Debido a que Cumbia es aplicable a diferentes dominios, surge la necesidad de tener un modelo extensible de definición de lenguajes que permita crear lenguajes de definición de procesos para Cumbia. De esta forma, para cada dominio se puede tener un lenguaje apropiado.

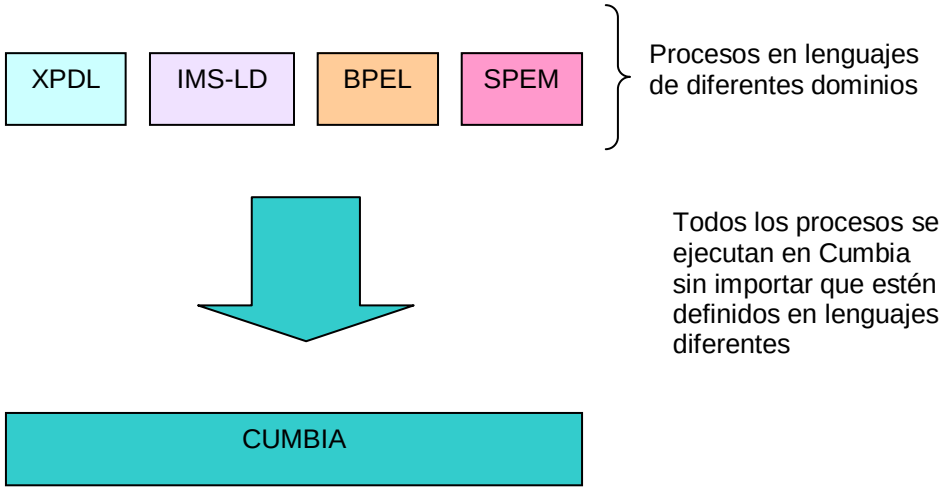

Figura No. 1: Procesos de diferentes dominios sobre Cumbia

Como se muestra en la anterior figura, se pueden tener procesos de diferentes dominios, los cuales se ejecutan en Cumbia. Para el usuario es transparente, es decir, el usuario define o diseña su proceso en el dominio que desee utilizando el lenguaje apropiado para tal dominio, y Cumbia es capaz de ejecutar el proceso.

Este proyecto propone un modelo de lenguaje extensible, llamado CUMBIA-XLM, que permite definir diferentes lenguajes gráficos para editar y visualizar procesos de Cumbia. Gracias a CUMBIA-XLM los usuarios pueden trabajar con Cumbia sin tener que aprender una notación grafica nueva sino que editan sus procesos con la notación grafica del dominio que conocen.

1.2. Objetivos

A continuación se describen el objetivo general y los objetivos específicos.

1.2.1. Objetivo General

Crear, especificar y evaluar un modelo extensible para definir lenguajes gráficos de definición de procesos para Cumbia.

1.2.2. Objetivos Específicos

- Identificar, definir y explicar cada uno de los elementos que hacen parte del modelo de definición de lenguajes.
- Explicar la extensibilidad del modelo para definir diferentes lenguajes gráficos.
- Crear una especificación del modelo extensible para definición de lenguajes gráficos.
- Establecer y explicar criterios para la evaluación del modelo.
- Evaluar el modelo con diferentes criterios, mediante la definición de lenguajes gráficos para algunos modelos de definición de procesos existentes actualmente en el mercado, los cuales son: WSFL, WS-BPEL, XPDL, SPEM, BPMN, IMS-LD.
- Diseñar e implementar la especificación para visualizar procesos de CUMBIA-XPM.

1.3. Estructura del Documento

El resto del documento está organizado por capítulos. El capítulo 2, presenta el contexto teórico de esta tesis. En el capítulo 3 se explica el modelo CUMBIA-XLM, el cual es el tema principal de este documento. El capítulo 4 contiene una evaluación del modelo CUMBIA-XLM, mostrando la forma en que se crean lenguajes con diferentes sintaxis gráficas. El capítulo 5 expone la arquitectura de Cumbia. En el capítulo 6 se pueden ver las conclusiones y el trabajo futuro.

2. Contexto

Este capítulo presenta un contexto teórico del proyecto. Primero se presenta una clasificación de los principales lenguajes, modelos y metamodelos existentes, que sirven para definir procesos. Luego, se presenta la visión global del proyecto Cumbia y se profundiza en el modelo CUMBIA-XPM. Por último, se exponen brevemente algunos proyectos de sistemas manejadores de *workflows* o de procesos de negocio.

2.1. Limitaciones de los Modelos Actuales

De acuerdo a [Sol05a], los modelos de definición de procesos actuales presentan una serie de limitaciones o desventajas. A continuación se enumeran y describen:

- **Lenguajes con poca capacidad de expresión.** La capacidad de expresión de un lenguaje de definición de procesos se mide principalmente por los patrones de workflow que se pueden expresar y la facilidad con que se hace. Existen muchos estudios en la literatura (algunos de ellos son: [Aal03], [ABHK00], [ADHW03], [AHKB00], y [WADH03]) que comparan diferentes lenguajes respecto a cuáles patrones de workflow pueden expresar. De estos estudios se llega a la conclusión que la mayoría de los lenguajes tienen poco poder de expresión, ya que no soportan muchos de los patrones de workflow existentes.

- **Ausencia de sintaxis gráfica.** La sintaxis gráfica facilita la construcción, el mantenimiento, y la comprensión de procesos de negocio; por tal razón es importante que los modelos de procesos posean una sintaxis gráfica. Sin embargo no todos los modelos poseen dicha sintaxis gráfica. Por ejemplo, las especificaciones de BPEL [ACDGKLLRSTTW03], jBPM [Bay05], XPDL [WMC01] no definen una sintaxis gráfica, sólo define una sintaxis textual.

- **Interacción limitada con otros modelos de procesos.** Esta interacción se puede dar por medio de la traducción entre lenguajes de definición de procesos o porque un motor de ejecución usa los servicios de otro motor. Por ejemplo, BPEL4WS no establece ningun tipo de traducción o mapeo a otros lenguajes. Por otro lado, existen mapeos o esquemas de traducción de diferentes lenguajes a XPDL, debido a que éste fue concebido como un lenguaje común de definición de procesos [WMC01]. En cuanto a que un motor utilice los servicios de otro, depende exclusivamente de la implementación.

- **Modelos poco extensibles.** Los mecanismos de extensión que proveen los modelos son difíciles de entender y usar, o no son suficientes para las exigencias que requiere modelar procesos de diferentes dominios [Sol05a]. Algunos mecanismos de extensión son muy pobres como el que ofrece XPDL, el cual consiste en la capacidad de definir atributos extendidos en la definición del proceso; pero los atributos extendidos deben ser soportados por la implementación del motor de ejecución, lo que vuelve al mecanismo de extensión dependiente de la implementación. Otros mecanismos pueden ser muy poderosos pero son dificiles o peligrosos de usar ya que pueden generar inconsistencias durante la ejecución de un proceso. Por ejemplo, APEL permite definir Aspectos, pero tanto definir un Aspecto como

depurarlo es una tarea compleja porque es difícil determinar en qué puntos se está afectando el comportamiento de un proceso.

- **Sistemas con dependencias tecnológicas.** Lo ideal es que existan modelos que se puedan implementar en cualquier tecnología y que la implementación de los modelos sea capaz de interactuar con sistemas externos sin importar la tecnología de estos. Sin embargo, esto no es lo que ocurre. Por ejemplo, BPEL4WS está orientado a interactuar solamente con Web Services; y jBPM sólo se puede utilizar con Java y J2EE [Sol05a].

Estas limitaciones forman el problema que Cumbia resuelve.

2.2. Cumbia

El proyecto Cumbia busca construir una plataforma tecnológica basada en modelos formales ejecutables, sobre la cual se puedan construir sistemas manejadores de procesos automatizados. Estos sistemas pueden estar enfocados en diferentes dominios.

Un punto clave es que los modelos de Cumbia están orientados a ser minimales y extensibles. Minimales porque incluyen los elementos mínimos necesarios para lograr los objetivos del modelo, y extensibles porque proveen las herramientas o mecanismos para extender el modelo. De esta forma, Cumbia soluciona las limitaciones que presentan actualmente los sistemas manejadores de procesos automatizados.

Cada modelo en los que se basa Cumbia aborda una problemática específica sobre los sistemas manejadores de procesos automatizados: Es así como existe el modelo CUMBIA-XPM el cual se enfoca en los elementos mínimos que se deben proveer para hacer posible la definición de procesos. El modelo CUMBIA-XRM, se enfoca en la asignación de recursos a las actividades de un proceso. El modelo CUMBIA-XLM, se enfoca en los lenguajes para diseñar y visualizar los procesos.

Cumbia no se limita a estos tres modelos, es decir, se pueden construir nuevos modelos que hagan que Cumbia sea más completo.

Para cada uno de estos modelos se implementan intérpretes reflexivos (llamados el motor del modelo), es decir que todos los elementos que existen en el modelo también existen en el motor, y viceversa.

Existen componentes que se encargan de la sincronización y comunicación entre los motores. Además de los componentes que se encargan de la administración.

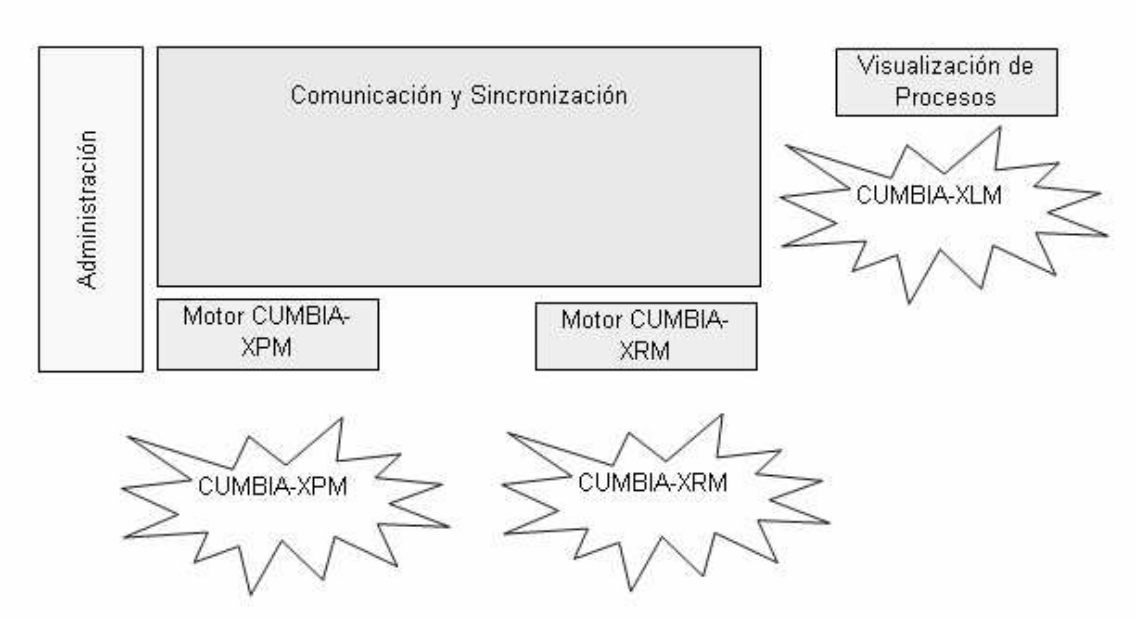

Figura No. 2: Cumbia

En la figura anterior se muestran los modelos CUMBIA-XPM y CUMBIA-XRM, sobre los cuales están basadas las implementaciones de los motores respectivos. El componente de Cumbia que hace posible la visualización de los procesos está basado en el modelo CUMBIA-XLM. El componente de comunicación y sincronización interactúa con los motores y con el visualizador de procesos.
Una aplicación que se ejecuta en Cumbia consiste en descriptores y datos dirigidos a cada modelo. Por ejemplo, para el modelo CUMBIA-XPM se debe especificar la definición del proceso; para el modelo CUMBIA-XRM se deben especificar los recursos existentes y las políticas de asignación de recursos; y para el modelo CUMBIA-XLM se debe especificar el lenguaje utilizado.

2.2.1. CUMBIA-XPM

CUMBIA-XPM es el modelo de definición de procesos de Cumbia [VTCNCSPM05]. Con este modelo se pueden definir los procesos como grafos dirigidos. Donde los nodos pueden ser actividades, multiactividades y procesos. Las transiciones de estos grafos son *dataflows*.

En la Figura No. 3, se muestra un ejemplo de un proceso con una actividad (“Citar evaluación”), una multiactividad (“Realizar evaluación”) y un proceso (“Seleccionar aspirantes”); y las transiciones entre cada nodo son los *dataflows*.

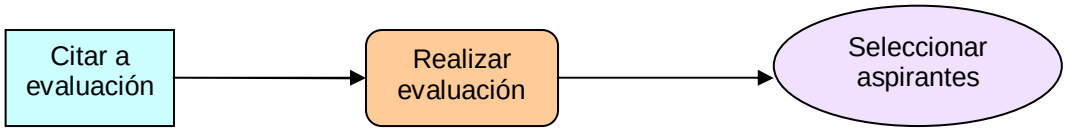

Figura No. 3: Ejemplo de un proceso

Una **actividad** es una tarea atómica que se debe realizar en un punto determinado dentro de un proceso. Por ejemplo, la actividad “Citar a evaluación” es el primer paso del proceso anterior.

Una **multiactividad** es una actividad con una característica especial, se ejecuta un número determinado de veces en forma concurrente. Por ejemplo, la actividad “Realizar una evaluación”, la cual debe ser realizada por todos las personas citadas.

Un **proceso** contiene actividades, mutiactividades y procesos. Por ejemplo, el proceso “Seleccionar aspirantes” contiene las actividades: “Calificar evaluaciones”, y “Seleccionar las mejores calificaciones” (Figura No. 4).

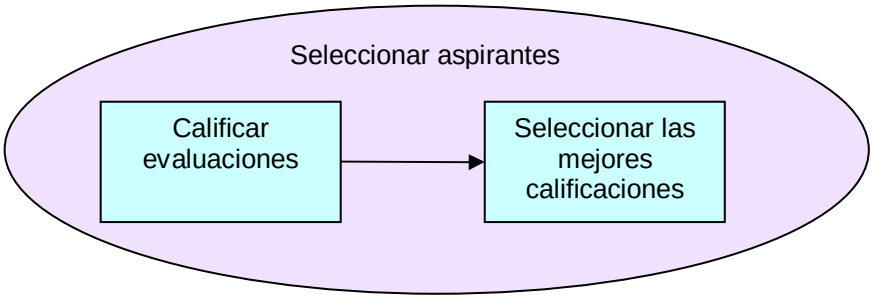

Figura No. 4: Proceso seleccionar aspirantes

Las actividades, multiactividades y procesos reciben datos, y arrojan otros datos al terminar su ejecución.

Para realizar la primera actividad del proceso, llamada “Citar a evaluación”, se cuenta con un sistema que consulta de una base de datos las direcciones de correo electrónico de las personas a citar, y les envía un correo electrónico con la citación. Por otro lado las demás actividades son realizadas por personas. En cualquiera de los dos casos las actividades y multiactividades necesitan de un mecanismo de comunicación con el mundo exterior, este es llamado *workspace*.

Un ***workspace*** es el elemento que está encargado de la comunicación entre una actividad o multiactividad y el mundo exterior, como por ejemplo una aplicación externa. Además, el *workspace* tiene la capacidad de consumir y producir datos, por tal razón está dividido en dos partes: un área llamada ***dataToProcess***, donde están los datos a procesar; y un área llamada ***dataProcessed***, donde están los datos procesados o producidos por el *workspace*. Un tipo de *workspace* puede tomar los datos del área *dataToProcess*, enviarlos a una aplicación externa, recibir datos de la aplicación externa, y poner los datos en el área *dataProcessed*. Otro tipo de *workspace* puede solicitar datos a través de un formulario en una página web y poner los datos en el área *dataProcessed*. Y así mismo puede haber otros tipos de *workspace* que tienen estrategias diferentes para procesar los datos.

Un **puerto** es un elemento que recibe datos y los mantiene hasta que otro elemento se los pide. La actividad, multiactividad y los procesos tienen uno o más puertos de entrada y salida. Un puerto de entrada recibe datos de un *dataflow* y los entrega a una

actividad, multiactividad o proceso. Un puerto de salida recibe datos de una actividad, multiactividad, o proceso y los entrega a un *dataflow*.

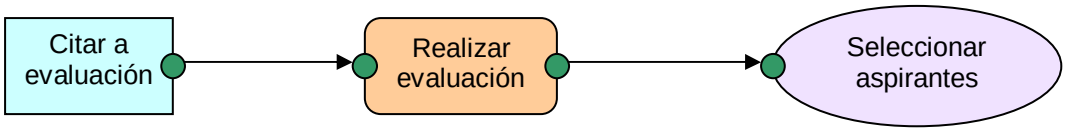

Figura No. 5: Ejemplo de un proceso con puertos

Un ***dataflow*** puede verse como un canal por donde fluyen los datos. Los *dataflows* deben tener un puerto origen y un puerto destino. El *dataflow* toma los datos del puerto origen y los envía al puerto destino. Además de transportar datos, un *dataflow* tiene la capacidad de realizar una operación de renombramiento de los datos.

Cada uno de estos elementos posee un espacio de memoria donde se mantienen los datos que procesa el elemento. Además, tienen asociado un autómata que define su comportamiento.

Los elementos que componen un autómata son: estados, transiciones, eventos y acciones.

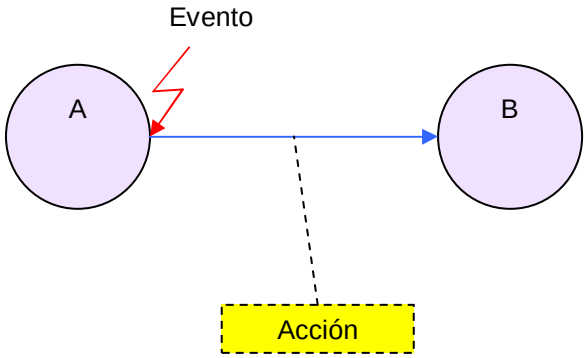

Figura No. 6: Ejemplo de autómata

La figura anterior es un ejemplo de un autómata. En él hay dos estados A y B, un evento, una transición de A hacia B, y una acción. El evento que se muestra permite la transición del autómata del estado A al estado B. Cuando el autómata toma la transición, la acción asociada se ejecuta.

Un **estado** se identifica con un nombre. El estado puede generar dos eventos, uno de ellos se genera cuando el autómata entra al estado y el otro se genera cuando el autómata sale del estado. Por ejemplo, en la siguiente figura, el evento E1 se genera cuando el autómata entra al estado A, y el evento E2 se genera cuando se sale del estado A.

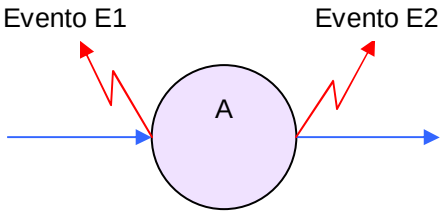

Figura No. 7: Eventos generados por un estado de un autómata

Una **transición** permite que el autómata cambie de un estado a otro. Un autómata toma una transición cuando ocurre un evento determinado. La transición genera dos eventos, uno antes de iniciar la transición y otro al salir de la transición. Además, una transición tiene asociado un conjunto de acciones que se ejecutan cuando el autómata toma la transición.

Un **evento** es una notificación que puede ser generada por un autómata cuando entra o sale de un estado, y cuando entra o sale de una transición. Las acciones también pueden generar eventos. Los eventos son escuchados por autómatas, y pueden causar la entrada a un estado o una transición.

Una **acción** básicamente es una clase en Java que implementa una interfaz con un método llamado *execute* el cual recibe un contexto como parámetro. El contexto es simplemente una referencia al espacio de memoria del elemento que ejecuta la acción. Las acciones pueden generar eventos.

La semántica del modelo se hace explícita mediante los autómatas de cada elemento. Un autómata genera eventos que hacen que otro autómata tome transiciones y ejecute las acciones asociadas a las transiciones. De esta forma la sincronización entre los elementos de un proceso es posible por los autómatas asociados a cada elemento.

Para explicar esto, considere el caso de un puerto conectado a un *dataflow*, de tal manera que el puerto juega el papel de puerto de entrada para el *dataflow*, ya que el *dataflow* obtiene los datos de dicho puerto. Tanto el puerto como el *dataflow* tienen sus propios autómatas.

En la siguiente figura se muestra una parte del autómata del puerto y del autómata del *dataflow*. Los números indican el comportamiento de los dos autómatas. A continuación se describe lo que ocurre en cada uno de estos pasos:

1. El autómata del puerto está en la transición que se dirige al estado *Waiting*, y el autómata del dataflow está en el estado *Inactive*.
2. El autómata del puerto, antes de entrar al estado *Waiting*, genera el evento *enterWaiting*. Este evento es escuchado por el autómata del dataflow, lo que hace que tome la transición del estado *Inactive* al estado *Empty*.
3. El autómata del puerto se encuentra en el estado *Waiting* y el autómata del dataflow en el estado *Empty*.
4. El autómata del puerto toma la transición del estado *Waiting* al estado *Full*, y cuando sale de ella genera el evento *enterFull.* El autómata del dataflow escucha este evento y toma la transición de *Empty* a *Full*. Al tomar esta transición se ejecuta la acción *getDataFromPort*, la cual toma los datos que están en el puerto.

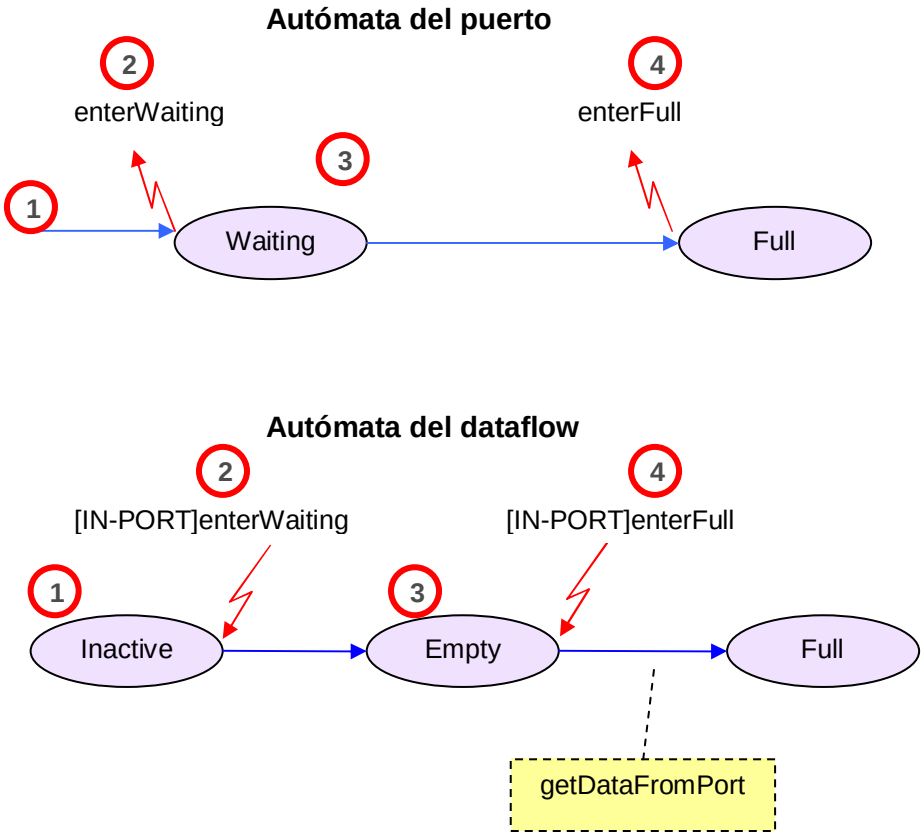

Figura No. 8: Sincronización entre los autómatas de un puerto y un dataflow

CUMBIA-XPM es un modelo minimal, es decir que tiene la menor cantidad de elementos posibles y provee un mecanismo de extensión para expresar las cosas que no se puedan hacer con los elementos que tiene el modelo.

La capacidad de extensión de CUMBIA-XPM es la característica que le da al modelo su gran poder expresivo. Según [Sol05b], con el modelo de procesos CUMBIA-XPM se pueden expresar 18 patrones de workflow, algunos de manera directa y otros usando algún mecanismo de extensión. Gracias al poder expresivo, este modelo es aplicable a problemas en diferentes dominios.

Como se dijo anteriormente, el poder expresivo del modelo se debe a los mecanismos de extensión. CUMBIA-XPM provee cuatro mecanismos de extensión: reflexión, extensión simple, adaptación, y especialización:

- La **reflexión** consiste en dos conceptos: el primero es la capacidad de conocer información de los elementos de un proceso en ejecución (introspección), y el segundo es la capacidad de modificar dicha información en tiempo de ejecución.

- La **extensión simple** consiste en incrementar la funcionalidad de un elemento agregando o eliminando acciones en las transiciones del autómata de dicho elemento. Las acciones se pueden agregar o eliminar sin cambiar la implementación de los elementos, ya que los autómatas están separados de la implementación de los elementos y las acciones son clases independientes.

- La **adaptación** consiste en cambiar el comportamiento de un elemento asignándole un autómata diferente. El nuevo autómata debe tener el mismo estado inicial que el autómata original, pero puede tener nuevos estados, transiciones y acciones. Por ejemplo, se pueden crear puertos o actividades con un comportamiento diferente y usarlos en nuevos procesos.

- La **especialización** en crear nuevos elementos extendiendo la funcionalidad básica de los elementos existentes. Esto se hace especializando la clase que implementa el elemento determinado, modificando los atributos o métodos.

Estos mecanismos de extensión permiten construir sistemas manejadores de procesos automatizados sobre Cumbia. En [Ped05] se explica la forma de representar BPEL4WS, XPDL y jBPM en Cumbia.

Básicamente, lo que se hace para definir un modelo X sobre CUMBIA-XPM es crear un conjunto Δx de nuevos elementos mediante los mecanismos de extensión. Así, Δx es un conjunto de elementos que pertenecen al modelo X, y que al mismo tiempo son elementos extendidos a partir de los elementos de CUMBIA-XPM y los mecanismos de extensión.

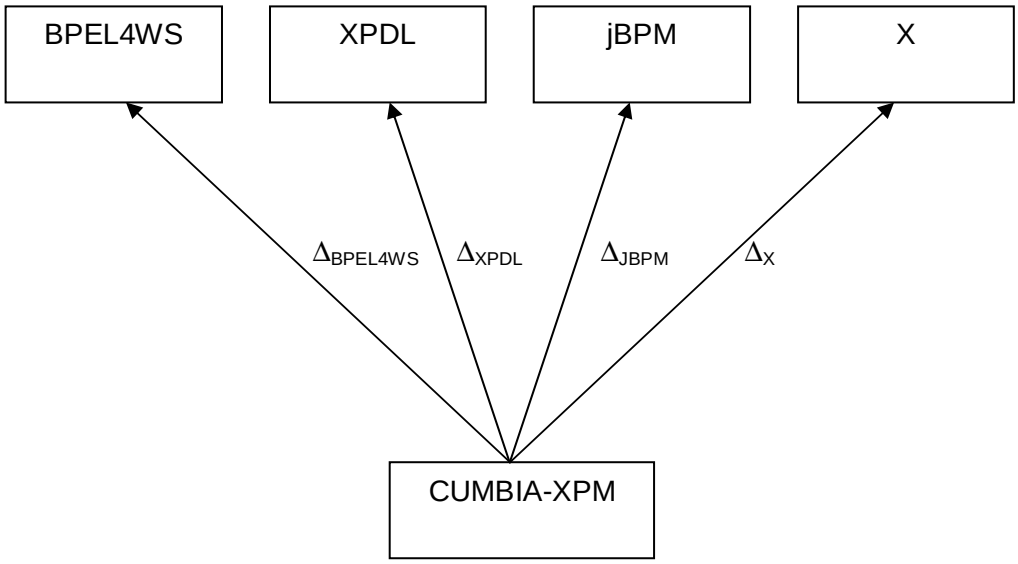

Figura No. 9: Extensiones del Modelo CUMBIA-XPM

2.2.2. CUMBIA-XRM

Un dominio importante en el ámbito de procesos es el dominio de recursos. En [RHEA05] se aborda el dominio de recursos en el contexto de procesos.

De acuerdo a [RHEA05] un recurso es una entidad capaz de llevar a cabo un trabajo. En esta definición, son considerados recursos: las personas, y las aplicaciones o sistemas.

La operación de asignar un recurso a una actividad o a un proceso consiste en determinar que la persona X o el sistema Y es el encargado de llevar a cabo el trabajo definido por la actividad o el proceso.

Existen muchos criterios o políticas que se tienen en cuenta para la asignación de un recurso. En [RHEA05], se proponen una serie de patrones de recursos.

Estos patrones se dividen en siete grupos:

- **Patrones de creación:** Sirven para delimitar el conjunto de recursos que pueden llevar a cabo una actividad.
- **Patrones push:** En este tipo de patrones el sistema de workflow toma la iniciativa de ofrecer las actividades que se crean a un conjunto de recursos.
- **Patrones pull:** Al contrario de los patrones push, el recurso toma la iniciativa de hacerse responsable de llevar a cabo una actividad.
- **Patrones detour:** Hacen referencia a situaciones donde las reservas de trabajo que se han hecho para recursos es interrumpida por el sistema de workflow o por algún recurso.
- **Patrones de auto-inicio:** Se refieren a situaciones donde una actividad inicia su ejecución automáticamente debido a un evento que ocurre. Dicha actividad debe tener un responsable.
- **Patrones de visibilidad:** Definen diferentes alcances en los cuales los recursos pueden acceder a las actividades.
- **Patrones de múltiples recursos:** Se refiere a situaciones en las que un recurso puede llevar a cabo varias tareas simultáneamente, y cuando una actividad pueden realizarla varios recursos al mismo tiempo.

El modelo CUMBIA-XRM aborda la problemática de asignar un recurso a una actividad o a un proceso, teniendo como marco de referencia los patrones de recursos.

CUMBIA-XRM provee un mecanismo de definición de reglas con las cuales se pretende dar soporte a las necesidades de asignación de recursos que se puedan presentar.

Algunos de los conceptos que maneja CUMBIA-XRM son: roles, jerarquías entre los recursos, habilidades y disponibilidad de los recursos, privilegios o permisos de un recurso para realizar una actividad, entre otros.

2.3. Otros Proyectos

En esta sección se pretende hacer una breve reseña de algunos proyectos de sistemas manejadores de *workflows* o de procesos de negocio; con el fin de dar una visión del contexto actual en esta área, e intentar identificar las limitaciones y fortalezas de estos respecto a Cumbia.

2.3.1. COSA

COSA es un sistema BPM (Business Process Management) que utiliza una base de datos relacional en un esquema cliente servidor. Este sistema se ha desarrollado cumpliendo con los estándares estipulados por la WfMC (Workflow Management Coalition).

Dos aspectos interesantes de COSA son que utiliza XPDL como lenguaje de definición de procesos y cumple con varios patrones de recursos [RHEA05].

Cumbia no tiene un lenguaje definido para los procesos, pero posee el modelo CUMBIA-XLM con el cual puede soportar diferentes lenguajes entre los cuales se encuentra XPDL. Además, CUMBIA-XRM, el modelo de recursos de Cumbia, hace posible el soporte de los patrones de recursos.

COSA está compuesto por los siguientes módulos: un modulo para el diseño de procesos (COSA Process Designer), un simulador que permite verificar la lógica del proceso en tiempo de modelación (COSA Simulator), un sistema administrador de las actividades de un proceso (COSA Worklist Handler), una herramienta de monitoreo (COSA Business Activity Monitor).

2.3.2. Intalio|n^3

Intalio|n^3 es un sistema manejador de procesos de negocio. Su arquitectura esta constituida por cuatro componentes: *server*, *designer*, *director*, y *projectors*.

Intalio|n^3 Server es el principal componente de la arquitectura y es el responsable de la ejecución de los procesos de negocio. Intalio|n^3 Designer permite importar procesos de negocio hechos en otras herramientas de software para ser ejecutados en el servidor de Intalio|n^3. El componente Intalio|n^3 Director permite que los usuarios puedan dirigir la ejecución de los procesos. Intalio|n^3 Projectors se encargan de exponer los sistemas externos como componentes de procesos para que puedan ser utilizados por estos.

El componente *server* contiene intérpretes para permitir la ejecución de procesos BPML y BPEL4WS. Con esta característica y gracias al componente Intalio|n^3 Designer que permite la importación de procesos diseñados en otros lenguajes es posible para Intalio|n^3 ejecutar procesos de negocio en diferentes lenguajes.

En pocas palabras, los intérpretes e importadores hacen parte de la estrategia de Intalio|n^3 para soportar diferentes lenguajes. Sin embargo, esta estrategia es simplemente a nivel tecnológico, lo que significa que para poder soportar un nuevo lenguaje es necesario implementar nuevos intérpretes e importadores.

La estrategia que propone Cumbia es a nivel de los modelos. Los modelos de Cumbia proveen los mecanismos necesarios para soportar diferentes lenguajes. Esta estrategia es mucho más elegante y menos costosa ya que no hay necesidad de implementar nuevos componentes como intérpretes.

2.3.3. Microsoft Windows Workflow Foundation (WWF)

Este es un framework extensible para el desarrollo y ejecución de *workflows* sobre la plataforma Windows.

El objetivo principal de este proyecto es proveer un motor para la ejecución de *workflows* con la capacidad de integrar e interactuar con todas las aplicaciones desarrolladas sobre la plataforma Windows.

Soporta *workflows* secuénciales y *workflows* basados en eventos. Los *workflows* secuénciales son auto conducidos y determinísticos. Los *workflows* basados en eventos son no determinísticos y son usados cuando personas intervienen en algún punto del *workflow*.

Por otro lado, Cumbia no hace diferenciación entre los procesos que ejecuta; sin embargo, Cumbia soporta procesos secuénciales, determinísticos y auto conducidos y procesos no determinísticos con intervención de personas.

WWF provee seis tipos de actividades que pueden ser utilizadas en los *workflows*. Estas son: actividades de control de flujo (por ejemplo: *sequence*, *parallel*, *if*, *while*), actividades de transacciones y excepciones (por ejemplo: *throw*, *compensate*, *suspend*), actividades basadas en la entrada de datos (por ejemplo: *updateData*, *waitForData*), actividades de comunicaciones (por ejemplo: *invokeWebService*, *webServiceReceive*), actividad de código, y actividades para el modelo máquina de estados.

Como se dijo anteriormente, WWF es un framework, por lo tanto no tiene un lenguaje de definición de *workflows*, simplemente provee un conjunto de APIs con las cuales se programan los *workflows*.

La arquitectura de WWF consiste en cuatro capas: la capa *workflow model*, la capa *runtime*, la capa *hosting*, y la capa *host process*.

La primer capa, la capa *workflow model* es la que permite que los desarrolladores construyan los *workflows* a partir de los elementos que provee WWF, como los tipos de actividades. La siguiente capa, la capa *runtime* es el núcleo de la arquitectura, porque provee los servicios críticos para la ejecución de *workflows*, como activación de actividades, manejo de eventos, excepciones, seguimiento, administración de estados, entre otros. La capa *hosting* provee los servicios de persistencia, comunicación, y transaccionalidad. Y la capa *host process* aloja el motor de ejecución de *workflows* de WWF.

2.3.4. Orchestra y Bonita

ObjectWeb (www.objectweb.org) esta trabajando en la integración de aplicaciones. Mediante un ESB (Enterprise Service Bus) ObjectWeb esta integrando sus componentes.

Entre estos componentes está Orchestra, el cual es un motor de orquestación de servicios web bajo BPEL y permite la integración con otros productos de ObjectWeb, como el servidor de aplicaciones Jonas. Orchestra provee un compilador BPEL, el cual se encarga de tomar la descripción de los procesos en BPEL y generar clases Java que se comprimen en un archivo JAR. El motor de ejecución de Orchestra recibe el

archivo JAR para iniciar la ejecución del proceso. Además, Orchestra provee herramientas de monitoreo y administración de los procesos que se encuentran en el motor de ejecución.

Otro componente de ObjectWeb es Bonita. Bonita es un sistema de *workflow* basado en J2EE que funciona sobre el servidor de aplicaciones JonAs. Congruente con las especificaciones de WfMC (The Workflow Management Coalition) y basado en un modelo de *workflow* propuesto por ECOO Team, el cual incorpora actividades anticipables como un mecanismo más flexible de ejecución de *workflows*.

Permite la definición de flujos de procesos, mediante grafos aciclicos dirigidos. Los nodos del grafo son las actividades. Un nodo también puede ser un subproceso.

Los procesos pueden ser de dos tipos: cooperativos, y modelos. En un proceso cooperativo varios usuarios participan en la misma instancia del proceso. Cuando se requiere una instancia del proceso por usuario se utilizan los procesos de tipo model. Por ejemplo, para un sistema de compras cada usuario necesita tener su propio contexto del proceso para que no se confundan los productos que van a comprar dos usuarios diferentes.

3. El Modelo CUMBIA-XLM

En este capítulo se explica el modelo CUMBIA-XLM, los elementos que lo componen y cómo están relacionados.

3.1. Visión Global de CUMBIA-XLM

CUMBIA-XLM es un modelo de lenguaje extensible con el cual se pueden definir diferentes lenguajes para el modelo CUMBIA-XPM. Estos lenguajes son gráficos y se espera que los usuarios puedan diseñar y ver en ejecución los procesos con estos lenguajes.

La idea general se muestra en la Figura No. 10. En ella se ve un usuario que diseña su proceso en un editor específico para XPDL. El resultado es una definición del proceso en XPDL. Esta definición de proceso es la entrada para un importador que traduce el proceso XPDL a Cumbia. El importador traduce el proceso original a un proceso descrito con el lenguaje definido en CUMBIA-XLM para XPDL. Esta nueva definición de proceso es la que Cumbia ejecuta y el usuario puede ver dicha ejecución a través de un visualizador. Este visualizador permite ver procesos definidos en diferentes lenguajes. El visualizador recibe notificaciones del motor de procesos de Cumbia, con las cuales refresca la visualización del proceso mostrando el estado de las actividades del proceso en ejecución.

Este mismo escenario se puede tener para otros dominios como BPEL, IMS-LD, SPEM, entre otros. De esta forma se debe tener un importador para cada diferente dominio.

Los importadores y el visualizador hacen parte de Cumbia, pero los editores son específicos de cada dominio, por ejemplo para XPDL se puede utilizar JaWE.

La ventaja de este enfoque es que cada usuario percibe que está trabajando sobre una plataforma específica para su dominio, pero en realidad todos los usuarios están trabajando sobre una misma plataforma, Cumbia. Esto permite tener, con poco esfuerzo, motores de ejecución de procesos para diferentes dominios.

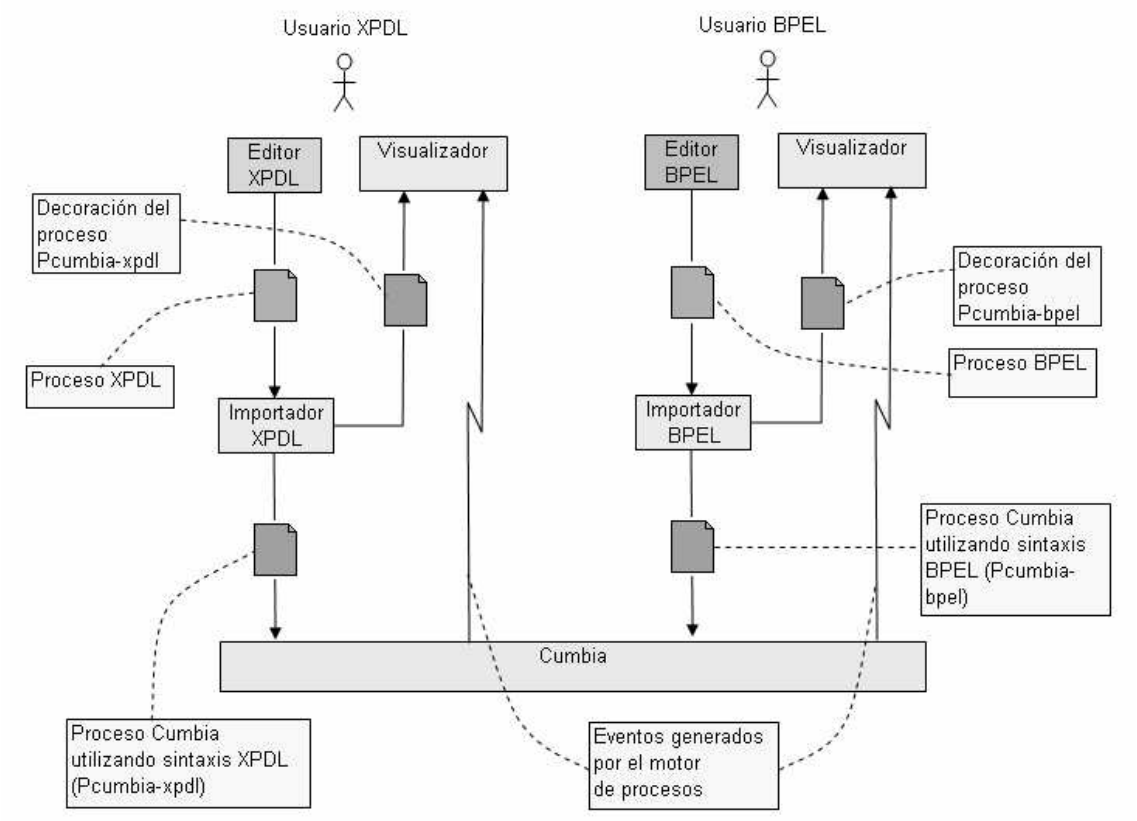

Figura No. 10: Idea general de CUMBIA-XLM

Las siguientes figuras muestran ejemplos de lenguajes que se pueden definir con CUMBIA-XLM.

En la Figura No. 11 se muestra un proceso con una sintaxis gráfica que asigna grafismos a los puertos de entrada y salida de los procesos y las actividades, pero no asigna grafismos para el *workspace* de las actividades o multiactividades.

En la Figura No. 12 se muestra un proceso con una sintaxis gráfica que asigna un grafismo al *workspace* de las actividades, tiene grafismos diferentes para la actividad y el proceso, tiene *dataflows* de dos colores diferentes, y no tiene asignado un grafismo para los puertos de entrada y salida.

El proceso de la Figura No. 13 muestra una sintaxis gráfica que permite agrupar las actividades y procesos de acuerdo al recurso o responsable de realizar dicha actividad o proceso. Además, tiene dos grafismos diferentes para las actividades.

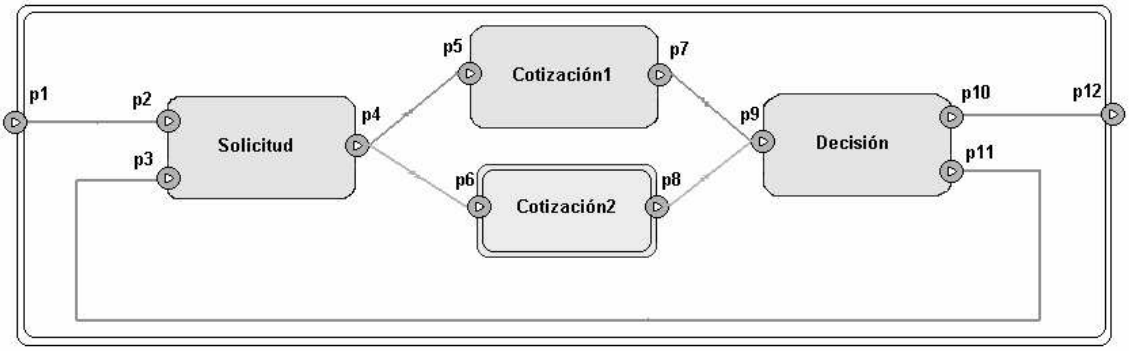

Figura No. 11: Ejemplo lenguajes 1

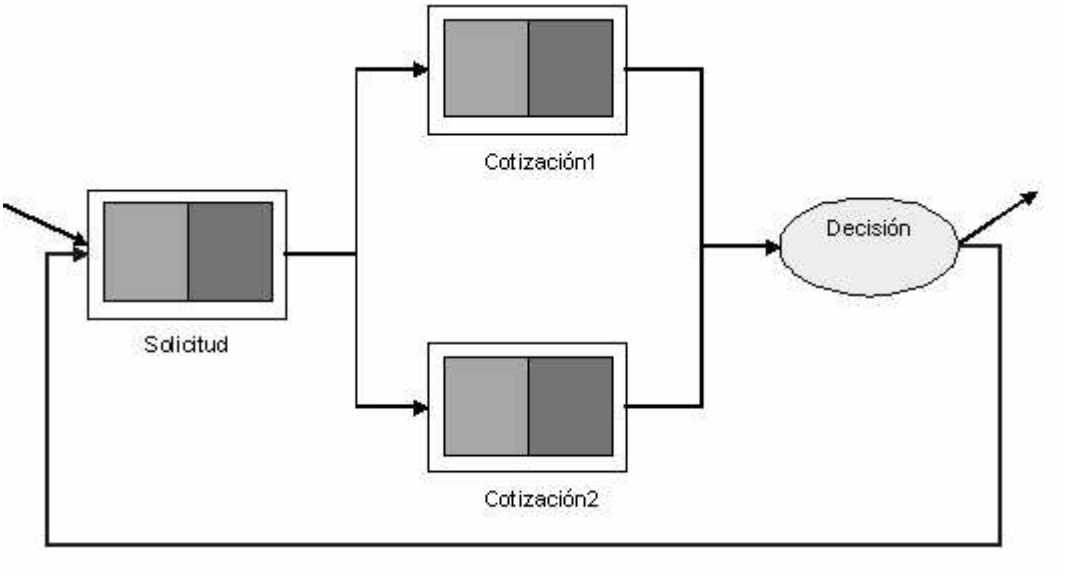

Figura No. 12: Ejemplo lenguajes 2

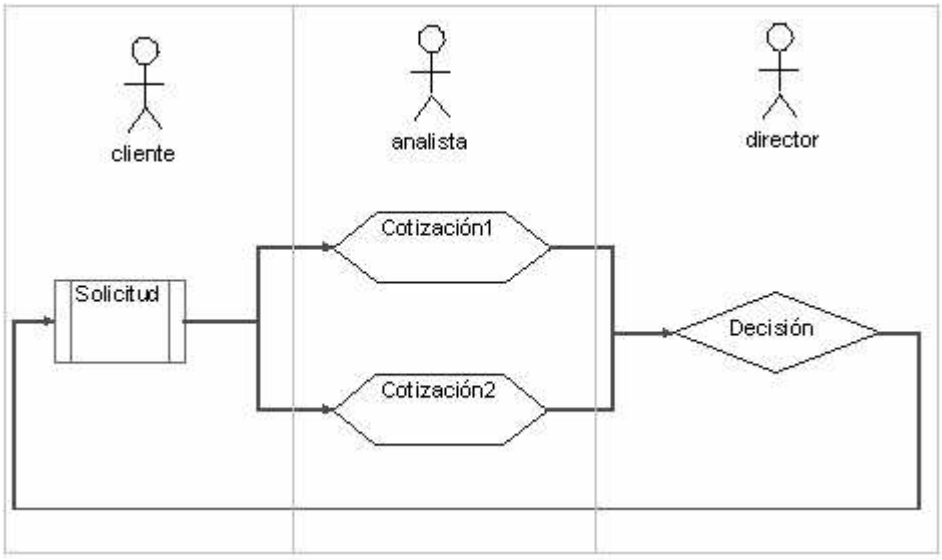

Figura No. 13: Ejemplo lenguajes 3

De estos ejemplos se puede ver que el modelo CUMBIA-XLM debe ser lo suficientemente flexible para asignar notaciones graficas a elementos de CUMBIA-XPM, o simplemente no asignarles notación grafica.

Los procesos de CUMBIA-XPM están definidos en documentos XML con tags predefinidos.

Con CUMBIA-XLM se tiene un escenario como el de la Figura No. 14, donde se tiene un proceso de CUMBIA-XPM definido en XML y se puede visualizar dicho proceso con diferentes grafismos sin modificar la definición del proceso CUMBIA-XPM.

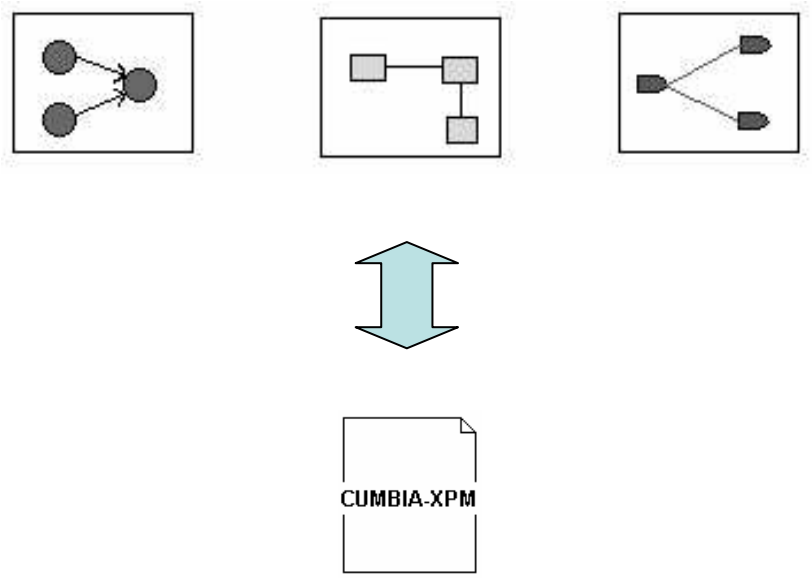

Figura No. 14: Visualización de un proceso

Para lograr esto se utiliza un documento XML que define el lenguaje gráfico, llamado estilo. De esta forma, con la definición del proceso CUMBIA-XPM y el estilo se puede visualizar el proceso de una forma determinada. Es decir, para cada forma de visualizar un proceso se debe definir un lenguaje gráfico diferente. Ese lenguaje gráfico se define en un documento XML.

En este mismo orden de ideas, se tiene el escenario de la Figura No. 15 donde se crean estilos para cada forma de visualización.

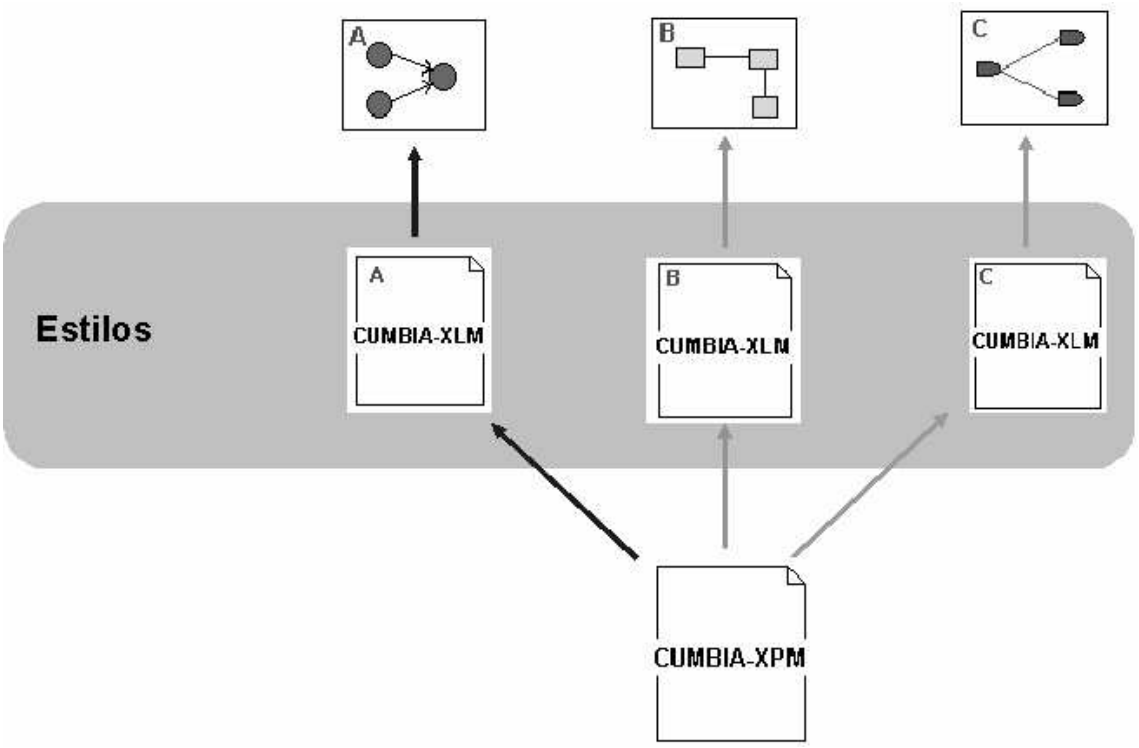

Figura No. 15: Estilos

Sin embargo, los estilos no son suficientes. Un aspecto que no se ha considerado son las posiciones o coordenadas de los elementos de un proceso a la hora de visualizar dicho proceso con un estilo. Hasta este momento no se ha especificado donde debe ir esta información.

Es claro que la información sobre coordenadas o ubicación de los elementos del proceso debe tenerse para visualizar el proceso de una forma ordenada y clara. Además, esta información es suministrada por el usuario cuando edita un proceso; y el usuario espera ver el proceso organizado de la misma forma como él lo organizó durante la edición.

Esta información no puede estar en el estilo, ya que el estilo sirve para definir un lenguaje gráfico que puede ser utilizado para visualizar diferentes procesos; tampoco debe ir en la definición del proceso porque se estaría contaminando el lenguaje textual en el que sólo deben ir elementos del modelo CUMBIA-XPM.

Por estas razones surge el concepto de decoración. Básicamente la decoración es un documento XML donde se especifica información que no se puede definir en el estilo ni en la definición del proceso en el lenguaje textual de CUMBIA-XPM, pero que se necesita para la visualización de un proceso determinado. Cabe resaltar que se debe definir una decoración por cada proceso.

Es así como se tiene el escenario de la Figura No. 16 donde se muestran tres elementos clave: la definición del proceso CUMBIA-XPM, la decoración asociada al proceso, y el estilo.

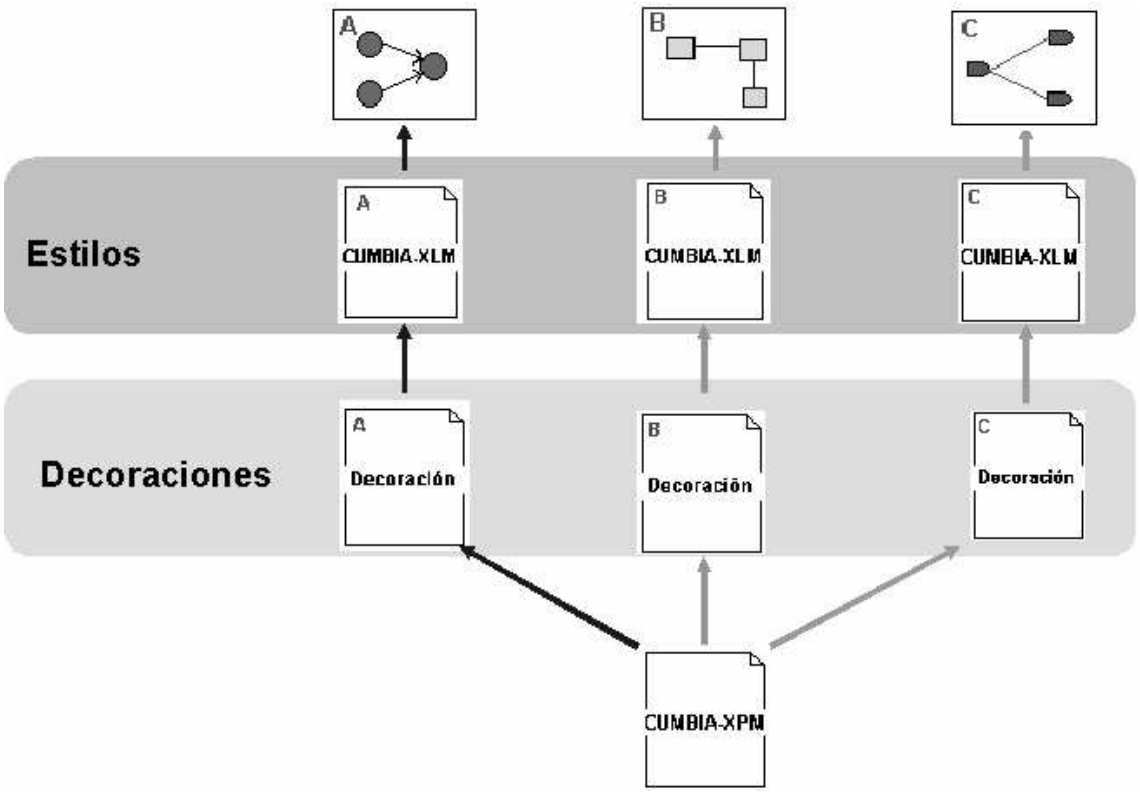

Figura No. 16: Estilos y decoraciones

3.2. Primera Vista al Estilo y la Decoración

Los conceptos de estilo y decoración son la base del modelo CUMBIA-XLM. Con un ejemplo se pueden ilustrar más detalladamente estos dos conceptos.

Suponga que queremos definir un lenguaje grafico en el que sólo existan actividades y *dataflows*. Las actividades se visualizan como un rectángulo y los *dataflows* como flechas.

Para definir la representación de las actividades se tienen los conceptos de *layout*, posición, y zona. Un *layout* contiene zonas, las cuales tienen una posición y un tamaño o dimensión especificas. En la siguiente figura se muestran las zonas que componen el *layout* de la actividad.

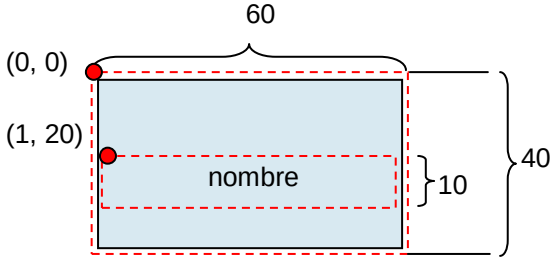

Figura No. 17: Zonas de una actividad

Las zonas están limitadas por los rectángulos punteados. La zona más grande tiene posición (x, y) igual a (0, 0) y dimensión ancho = 60, alto = 40. Esta zona tiene un rectángulo con borde y relleno con un color. La zona más pequeña tiene posición (1, 20), ancho = 60 y alto = 10. Esta zona contiene el nombre de la actividad.

La parte del estilo que define la representación de la actividad de la Figura No. 17 se muestra en la Figura No. 18. Como se ve, el estilo se define en XML.

El elemento *representation* sirve para asignar una representación grafica a un elemento del modelo CUMBIA-XPM. Tiene un elemento *layout* el cual sirve para describir elementos gráficos y textuales que se desea que sean visibles.

Un layout se especifica mediante un área rectangular, la cual tiene un ancho y un alto definidos por el tag *dimension*. Esta área rectangular se divide en subareas rectangulares llamadas zonas. Para definir una zona se necesitan coordenadas X y Y (tag *position*), las cuales son relativas al área del layout, y un ancho y un alto (tag *dimension*).

En una zona se pueden ubicar elementos gráficos o textuales, por ejemplo, en la representación de la actividad en el estilo anterior, hay una zona que contiene una figura geométrica (tag *geometry*) y otra zona que contiene un elemento textual que corresponde al nombre de la actividad (tag *subconcept*).

El elemento *geometry* tiene los atributos: *shape* que indica la figura geométrica (‘rectangle’, ‘roundrectangle’, ‘oval’) que tiene la zona; el atributo *fill* que indica si la figura se rellena o no con un color; y el atributo *border* que indica si la figura tiene borde o no. Además, el elemento *geometry* contiene un elemento *color* el cual especifica el color de la figura geométrica.

```
<representation concept="activity">
  <layout>
    <dimension width="60" height="40"/>
    <position-zone>
      <position x="0" y="0"/>
      <zone>
        <dimension width="60" height="40"/>
        <form>
          <geometry shape="rectangle" fill="true" border="true">
            <color r="216" g="233" b="241" alfa="200"/>
          </geometry>
        </form>
      </zone>
    </position-zone>
    <position-zone>
      <position x="1" y="20" />
      <zone>
        <dimension width="40" height="10"/>
        <subconcept>name</subconcept>
      </zone>
    </position-zone>
  </layout>
</representation>
```

Figura No. 18: Ejemplo de representación de un estilo

Otro elemento que se quiere definir en el lenguaje del ejemplo al inicio de esta sección es el *dataflow*. Esto se hace en el estilo mediante la representación del *dataflow* (tag *dataflow-representation*). Este elemento tiene atributos que caracterizan la forma de visualizar un *dataflow*, tales como el patrón de línea (*solid* o *dashed*), el estilo de línea (*squared* o *rounded*), el ancho de la línea, el tipo de dirección al inicio y final de la línea. Además tiene un color especificado en formato RGB con transparencia (atributo *alfa*).

```
<dataflow-representation line-pattern="solid" line-style="squared"
                         line-width="3" begin-direction-type="none"
                         end-direction-type="simple">
    <color r="0" g="0" b="0" alfa="200"/>
</dataflow-representation>
```

Figura No. 19: Ejemplo de representación de un dataflow

Uniendo las representaciones definidas en la Figura No. 18 y en la Figura No. 19 se tiene el estilo para el lenguaje del ejemplo. Esto se hace mediante el tag *style*, el cual tiene un atributo llamado *name* para darle al estilo un nombre.

```
<style name="ejemplo">

   <representation concept="activity">
      ...
   </representation>

   <dataflow-representation ...>

   </dataflow-representation>

</style>
```

Figura No. 20: Estructura de un estilo en XML

Aunque este estilo no tiene todos los elementos que provee CUMBIA-XLM, muestra dos elementos importantes: *representation* y *dataflow-representation*.

El otro concepto que se quiere ilustrar en esta sección es la decoración. Para esto, considere un proceso que sólo contiene una actividad, en la Figura No. 21 se muestra la definición de este proceso.

```
<process-definition>
  <process name="root"> <!-- Proceso raiz -->

    <activity name="A"> <!-- Actividad A -->
      <ports type="entry"><!--Puertos de entrada de la actividad-->
        <port name="e1">
          <variables>
            <variable name="a"/>
          </variables>
        </port>
      </ports>
      <ports type="exit"> <!-- Puertos de salida de la actividad-->
        <port name="s1">
          <variables>
            <variable name="b"/>
          </variables>
        </port>
      </ports>
      <workspace class="uniandes.cumbia.WorkspaceWS"/>
    </activity>

    <ports type="entry"><!-- Puertos de entrada del proceso raiz-->
      <port name="inicio">
        <variables>
          <variable name="a"/>
        </variables>
      </port>
    </ports>
    <ports type="exit"> <!-- Puertos de salida del proceso raiz-->
      <port name="fin">
        <variables>
          <variable name="b"/>
        </variables>
      </port>
    </ports>

    <dataflow from="inicio" to="A.e1">
      <map to="a" from="a"/>
    </dataflow>
    <dataflow from="A.s1" to="fin">
      <map to="b" from="b"/>
    </dataflow>

  </process>
</process-definition>
```

Figura No. 21: Definición de un proceso

Un documento XML con la decoración de este proceso se muestra en la Figura No. 22.

```
<decoration-process-definition>
  <process name="root"> <!-- Proceso raiz -->

    <activity name="A" x="10" y="10"> <!-- Actividad A -->
      <ports type="entry"><!--Puertos de entrada de la actividad-->
        <port name="e1"/>
      </ports>
      <ports type="exit"> <!-- Puertos de salida de la actividad-->
        <port name="s1"/>
      </ports>
    </activity>

    <ports type="entry"><!-- Puertos de entrada del proceso raiz-->
      <port name="inicio" x="5" y="10"/>
    </ports>
    <ports type="exit"> <!-- Puertos de salida del proceso raiz-->
      <port name="fin" x="50" y="10"/>
    </ports>

    <dataflow from="inicio" to="A.e1">
      <point x="5" y="10"/>
      <point x="10" y="15"/>
    </dataflow>
    <dataflow from="A.s1" to="fin">
      <point x="30" y="15"/>
      <point x="50" y="10"/>
    </dataflow>

  </process>
</decoration-process-definition>
```

Figura No. 22: Decoración de un proceso

Este ejemplo de decoración sólo contiene las coordenadas de los elementos del proceso (resaltadas en la figura anterior), aunque puede ir otro tipo de información que se explica más adelante.

3.3. Vista Detallada al Estilo

3.3.1. Representaciones gráficas

Un estilo está compuesto por representaciones gráficas. En el ejemplo de la sección anterior se mostraron dos tipos de representaciones: una representación de una actividad que tenía un *layout* y otra representación para los *dataflows*.

Las representaciones que tienen un layout se pueden utilizar para asignar notaciones gráficas a elementos de CUMBIA-XPM, tales como: actividad, multiactividad, proceso, puerto, entre otros.

La representación del *dataflow* sólo sirve para caracterizar la forma de visualizar un *dataflow*.

Existe otro tipo de representación especial que sirve para caracterizar la forma de visualizar asociaciones entre elementos en un proceso. En la sección 3.5, que habla de los mecanismos de extensión, se profundiza más acerca de esto.

3.3.2. Elementos con representaciones gráficas

Es importante tener identificados los elementos de CUMBIA-XPM que pueden tener una o más representaciones en un estilo. Estos elementos son: dato, esquema de traducción de los *dataflows*, puerto de entrada, puerto de salida, *workspace* y sus partes *dataToProcess* y *dataProcessed*, recurso, actividad, multiactividad y proceso.

Los elementos mencionados se denominan conceptos en CUMBIA-XLM, por tal razón en la definición de una representación en XML el tag *representation* tiene un atributo *concept* y el valor de este atributo puede ser cualquiera de los conceptos arriba mencionados.

3.3.3. Zonas que hacen referencia a figuras o imágenes

Las zonas pueden hacer referencia a una figura geométrica o a una imagen. Las figuras geométricas pueden ser: rectángulo, rectángulo redondeado, y óvalo. La imagen puede ser cualquier archivo de imagen en formato JPEG, PNG o GIF.

La siguiente figura muestra un ejemplo de una zona que hace referencia a una figura geométrica.

```
<zone>
  <dimension width="40" height="40"/>
  <form>
    <geometry shape="oval" fill="true" border="true">
      <color r="216" g="233" b="241" alfa="200"/>
    </geometry>
  </form>
</zone>
```

Figura No. 23: Zona referenciando una geometría

La siguiente figura muestra un ejemplo de una zona que hace referencia a una imagen.

```
<zone>
  <dimension width="40" height="40"/>
  <form>
    <image file="img.gif"/>
  </form>
</zone>
```

Figura No. 24: Zona referenciando una imagen

3.3.4. Zonas que hacen referencia a información textual

Además de las imágenes y las figuras geométricas, las zonas pueden contener información textual como el nombre de una actividad, el valor de un dato, el nombre de un recurso, entre otros.

Se puede decir que hay tres tipos de información textual que pueden agregarse en una zona.

Subconcepto

El primer tipo se llama subconcepto. Los subconceptos representan datos que pertenecen a los conceptos. Estos son:

- **Nombre:** nombre de un elemento de CUMBIA-XPM. Este subconcepto pertenece a los conceptos actividad, multiactividad, proceso, puerto, y recurso.
- **Variable:** nombre de una variable. Este subconcepto pertenece a los conceptos puerto, *dataflow* y *workspace*.
- **Value:** valor de un dato. Este subconcepto pertenece a los conceptos puerto, *dataflow* y *workspace*.
- **To:** nombre de una variable origen en el esquema de traducción de un *dataflow*
- **From:** nombre de una variable destino en el esquema de traducción de un *dataflow*.
- **nInstances:** número de instancias de una multiactividad.
- **resourceName:** nombre de un recurso.

```
<zone>
  <dimension width="40" height="20"/>
  <subconcept>name</subconcept>
</zone>
```

Figura No. 25: Zona que referencia un subconcepto

Label

El segundo tipo se llama label, el cual es una etiqueta o un valor constante que se especifica en la zona por medio del tag *label*.

```
<zone>
  <dimension width="80" height="20"/>
  <label>Actividad</label>
</zone>
```

Figura No. 26: Zona que contiene un label

Textfield

El tercer tipo se llama *textfield*. La existencia de este elemento se justifica con la extensibilidad del modelo, es por esto que este elemento se explica en la sección de los mecanismos de extensión de CUMBIA-XLM (sección 3.5).

Por ahora sólo se muestra cómo se incluye un *textfield* en una zona.

```
<zone>
  <dimension width="80" height="20"/>
  <textfield id="texto1"/>
</zone>
```

Figura No. 27: Zona que contiene un textfield

3.3.5. Zonas que hacen referencia a una representación

Las zonas también pueden hacer referencia a otras representaciones definidas en el mismo estilo. Esta característica es útil para crear representaciones de conceptos que contienen otros conceptos.

Por ejemplo, dado que el concepto actividad contiene al concepto *workspace*, se puede crear una representación para el *workspace* y otra representación para la actividad, la cual tiene una zona que hace referencia a la representación del *workspace*.

La siguiente figura ilustra la situación presente en el ejemplo. Primero se define una representación para el esta información. Esta representación contiene una zona con un óvalo. Luego, se define la representación de la actividad. La primera zona de esta representación contiene un rectángulo. La segunda zona es la que hace referencia a la representación del *workspace*, mediante el *tag representation-ref*. La tercera zona hace referencia al nombre de la actividad.

```
<representation concept="workspace">
  <layout>
    <dimension width="10" height="10"/>
    <position-zone>
      <position x="0" y="0"/>
      <zone>
        <dimension width="40" height="40"/>
        <form>
          <geometry shape="oval" fill="true" border="true">
            <color r="216" g="2" b="2" alfa="100"/>
          </geometry>
        </form>
      </zone>
    </position-zone>
  </layout>
</representation>

<representation concept="activity">
  <layout>
    <dimension width="40" height="40"/>
    <position-zone>
      <position x="40" y="0"/>
      <zone>
        <dimension width="40" height="40"/>
        <form>
          <geometry shape="rectangle" fill="true" border="true">
            <color r="216" g="233" b="241" alfa="200"/>
          </geometry>
        </form>
      </zone>
    </position-zone>
    <position-zone>
      <position x="0" y="20" />
      <zone>
        <dimension width="40" height="20"/>
        <representation-ref>workspace</representation-ref>
      </zone>
    </position-zone>
    <position-zone>
      <position x="0" y="20" />
      <zone>
        <dimension width="40" height="20"/>
        <subconcept>name</subconcept>
      </zone>
    </position-zone>
  </layout>
</representation>
```

Figura No. 28: Zona referenciando una representación

Existen otras situaciones en las que se puede utilizar la idea de hacer referencia a representaciones en las zonas:

- Por ejemplo, el *workspace* contiene dos conceptos que se llaman *dataToProcess* y *dataProcessed*. Por lo tanto, se puede definir una representación para el *workspace* con una zona que referencia a la representación de *dataToProcess* y otra zona que referencia a la

representación de dataProcessed. Desde luego, las representaciones para dataToProcess y dataProcessed deben estar definidas en el mismo estilo.

- Además del workspace, una actividad también contiene puertos de entrada y salida. Para esta situación, se pueden tener representaciones para puertos de entrada y salida y desde la representación de la actividad hacer referencia a las representaciones de los puertos de entrada y salida mediante las zonas.

En este último escenario surge un problema. Hasta ahora en cada zona se incluye un solo elemento, que puede ser grafico, textual u otra representación. Esto no presenta ningún problema de ubicación ya que la zona tiene una posición, la cual se usa para ubicar el elemento al que se hace referencia en dicha zona. Sin embargo, cuando en una zona de la representación de la actividad, multiactividad o proceso se hace referencia a un puerto de entrada o de salida, no ocurre lo mismo; ya que el número de puertos de entrada y salida no son predefinidos. Es decir, que puede haber zonas que hacen referencia a varios elementos del mismo tipo y no se sabe cómo ubicar cada uno de estos elementos en la zona. La posición de la zona no es suficiente para ubicar más de un elemento en una zona. Por tal razón se crean atributos de ubicación para las zonas.

3.3.6. Atributos de las zonas

Las zonas pueden contener atributos que asisten en la ubicación de elementos en la zona.

Existen dos conjuntos de atributos. El primer conjunto de atributos se utiliza en zonas que contienen más de un elemento, y el segundo conjunto de atributos se utiliza en zonas que sólo contienen un elemento.

Primer conjunto de atributos

Los atributos del primer conjunto son:

- **Initx:** Coordenada X inicial a partir de la cual se empiezan a ubicar los elementos de la zona.
- **Inity:** Coordenada Y inicial a partir de la cual se empiezan a ubicar los elementos de la zona.
- **Deltax:** Distancia en el eje X que especifica la separación entre los elementos de la zona.
- **Deltay:** Distancia en el eje Y que especifica la separación entre los elementos de la zona.

El siguiente es un ejemplo de una zona que hace referencia a puertos de entrada (*entryport*). Debido a que esta zona puede contener más de un puerto de entrada, se utiliza el primer conjunto de atributos (tag *attributes1*).

```
<zone>
  <dimension width="40" height="40"/>
  <representation-ref>entryport</representation-ref>
  <attributes1 initx="0" inity="1" deltax="0" deltay="2"/>
</zone>
```

Figura No. 29: Zona con atributos 1

Si se desea ubicar tres puertos de entrada en la zona, las coordenadas (X, Y) de cada puerto son: (0, 1), (0, 1 + Hport + 2), y (0, 1 + Hport + 2 + Hport + 2) respectivamente; donde Hport es la altura de un puerto de entrada.

Las coordenadas del primer puerto son (0, 1) porque así se especifica en los atributos initx e inity de la zona. Las coordenadas del segundo puerto son (0, 1 + Hport + 2); X = 0 porque el atributo deltax es igual a cero, y para obtener la coordenada Y se suman: la coordenada Y del puerto anterior (1), la altura del puerto (Hport), y el valor del atributo deltay (2). Las coordenadas del tercer puerto se calculan de forma similar.

Si el valor de deltax es un valor diferente a cero, como por ejemplo 3, entonces las coordenadas en X para los puertos serían: 0, 0 + Wport + 3, y 0 + Wport + 3 + Wport + 3.

En forma general, teniendo los elementos de una zona ordenados, siendo el primer elemento el elemento E_0, el segundo E_1, y así sucesivamente. Y suponiendo que W_E y H_E son el ancho y el alto respectivo de la representación de los elementos, las coordenadas de estos elementos en la zona se calculan de la siguiente manera.

$$E_0 = \begin{cases} X_0 = initx \\ Y_0 = inity \end{cases}$$

$$E_N = \begin{cases} X_N = \begin{cases} 0, \text{ si } deltax = 0 \\ X_{N-1} + W_E + deltax \end{cases} \\ Y_N = \begin{cases} 0, \text{ si } deltay = 0 \\ Y_{N-1} + H_E + deltay \end{cases} \end{cases}$$

Es claro que estos atributos pertenecen a una zona. La zona tiene una dimensión que especifica un ancho y un alto. Puede suceder que el número de elementos que se desean ingresar en la zona no quepan dados la dimensión de la zona, y los valores para los atributos de la zona. Por ejemplo, una zona puede tener como dimensión ancho = 20 y alto = 40 y los valores de sus atributos intix = 0, inity = 0 deltax = 0, deltay = 5. Supongamos que esta zona contiene tres elementos con ancho = 10 y alto = 20. Es claro que tres elementos de alto = 20 no caben en la zona de alto = 40.

Debido a situaciones como la del ejemplo anterior, las zonas tienen la capacidad de escalar las veces que sea necesario para poder contener los elementos que se desean. Sin embargo, para que la simetría de las representaciones que se definen en el estilo se conserve, el factor de escalado se aplica no sólo a una zona sino a todas las zonas de una representación.

Segundo conjunto de atributos

Este conjunto de atributos se usa en las zonas que contienen sólo un elemento grafico o textual.

Estos atributos son:

- **Halign:** Alineación horizontal del elemento dentro de la zona. Los posibles valores son: left, center, right.
- **Valign:** Alineación vertical del elemento dentro de la zona. Los posibles valores son: top, center, bottom.
- **Margin-top:** Define un espacio entre la parte superior de la zona y el elemento contenido por la zona.
- **Margin-bottom:** Define un espacio entre la parte inferior de la zona y el elemento contenido por la zona.
- **Margin-left:** Define un espacio entre el lado izquierdo de la zona y el elemento contenido por la zona.
- **Margin-right:** Define un espacio entre el lado derecho de la zona y el elemento contenido por la zona.

De acuerdo con estos atributos, una zona se puede ver como un área rectangular subdividida en las partes que se muestran en la siguiente figura.

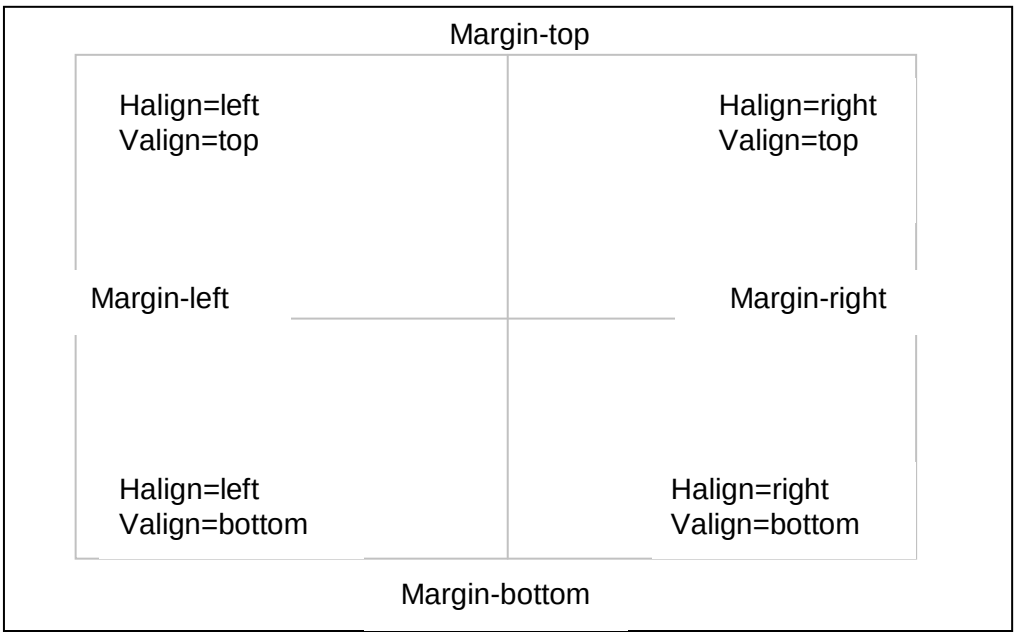

Figura No. 30: Atributos de una zona

3.3.7. Escalado de figuras e imágenes

Como se explico anteriormente, el tamaño de las zonas puede escalar cuando se requiere. Si la zona contiene una imagen o una figura geométrica, el estilo cuenta con la posibilidad de especificar si se quiere que esa imagen o figura escale o no como escala la zona. Esto se hace mediante el tag *form* el cual tiene un atributo llamado *scale* que puede tomar los valores 'true' o 'false'.

3.4. Mecanismos de Visualización de un Proceso en Ejecución

El modelo CUMBIA-XLM posee unas características que permiten ver un proceso en ejecución. Es decir, que se pueden ver cosas como: cambios de estado en los elementos del proceso, actividades que se están ejecutando, y datos que tiene un puerto o un *workspace*.

Para poder visualizar los cambios de estado en los elementos de un proceso se debe especificar una representación en el estilo para cada estado de cada elemento que nos interese ver. Es por esto que una representación tiene un atributo estado con el cual se le puede asignar un estado a la representación.

Por ejemplo, si queremos diferenciar entre una actividad que se esta ejecutando y una que no se está ejecutando, podemos definir dos representaciones: una para el estado de la actividad cuando no se ejecuta y otra para el estado de la actividad cuando se ejecuta.

```
<!-- actividad cuando se ejecuta -->
<representation concept="activity" state="active">
   ...
</representation>

<!--actividad en cualquier otro estado -->
<representation concept="activity">
   ...
</representation>
```

Figura No. 31: Manejo de estados en las representaciones

En la figura anterior, la primera representación tiene en su atributo *state* el estado *active*, esto indica que esa es la representación de la actividad cuando se está ejecutando. La segunda representación no tiene un estado asignado, esto indica que para cualquier otro estado de la actividad diferente a *active* se toma esta representación.

Es importante tener en cuenta que los estados que se asignan a las representaciones son los estados que pueden tomar cada elemento. Los estados de cada elemento están definidos en los autómatas de cada elemento, que se explican en la especificación del modelo CUMBIA-XPM [VTCNCSPM05].

Como los *dataflows* también tienen diferentes estados, la representación de los *dataflows* también tiene un atributo estado donde se le puede asignar un estado a una representación de un *dataflow*, de la misma forma que a la representación de la actividad en el ejemplo anterior.

```
<dataflow-representation state="inactive" ...>
   . . .
</dataflow-representation>
```

Figura No. 32: Manejo de estados en las representaciones de un dataflow

Para ver los datos que tiene un puerto o un *workspace* en determinado momento de la ejecución de un proceso, se cuenta con un mecanismo de interacción con el usuario que permite especificar qué información mostrar, cómo mostrarla y cuándo mostrarla.

El cuando mostrar la información se especifica haciendo referencia a los eventos del ratón que genera el usuario. Estos eventos del ratón son: clic, doble clic, clic derecho. Por ejemplo, si el usuario hace clic derecho sobre un puerto se puede abrir una ventana mostrando los datos que tiene el puerto en ese momento de la ejecución.

Para esto, el estilo cuenta con el elemento *actionListener* donde se especifica el evento por medio del atributo *event*. El elemento actionListener contiene un elemento *window* que especifica que la información se debe mostrar en una ventana emergente. El elemento *window* puede contener un *layout*, el cual dice cómo y qué información mostrar.

Además el elemento *window* tiene los atributos *modal, type,* y *title*.

El atributo *modal* indica si la ventana es bloqueante o no. Si una ventana es bloqueante el usuario no puede acceder a otras ventanas de la aplicación. Si modal es true la ventana es bloqueante, si es false la ventana no es bloqueante.

El atributo *type* indica el tipo de ventana. Los posibles valores que puede tomar son: ‘showInfo’ y ‘showProcess’. El tipo ‘showInfo’ quiere decir que la ventana se utilizará para mostrar información textual. El tipo ‘showProcess’ quiere decir que la ventana se utilizará para mostrar un subproceso.

El atributo *title* contiene el titulo de la ventana.

Para ilustrar mejor este concepto veamos como se define el *actionListener* para la representación de un puerto de entrada, de tal forma que cuando se haga clic derecho sobre un puerto de entrada se muestren los datos del puerto.

```
<representation concept="entryport" state="active">
   <layout>
      . . .
   </layout>
   <action-listener event="mouseRightClicked">
      <window modal="true" type="showInfo" title="Variables">
         <layout>
            <dimension height="100" width="150"/>
            <position-zone>
               <position x="5" y="0"/>
               <zone>
                  <label>Datos</label>
                  <dimension width="150" height="20"/>
                  <attributes2 halign="left" valign="top"
                               margin-top="0" margin-bottom="0"
                               margin-right="0"
                               margin-left="0"/>
               </zone>
            </position-zone>
            <position-zone>
               <position x="0" y="20"/>
               <zone>
                  <representation-ref>data</representation-ref>
                  <dimension width="150" height="80"/>
                  <attributes1 initx="2" inity="0" deltax="0"
                               deltay="1"/>
               </zone>
            </position-zone>
         </layout>
      </window>
   </action-listener>
</representation>
```

Figura No. 33: ActionListener para un puerto

Cuando se visualiza un proceso, éste puede tener subprocesos. Para visualizar las actividades de un subproceso se puede definir un *actionListener* para la representación del proceso de tal forma que al hacer doble clic sobre un subproceso se despliegue una ventana mostrando las actividades del subproceso. La siguiente figura muestra esta definición en el estilo.

```
<representation concept="process">
   <layout>
      . . .
   </layout>
   <action-listener event="mouseDoubleClicked">
      <window modal="true" type="showProcess" title="Subprocess">
   </action-listener>
</representation>
```

Figura No. 34: ActionListener para un proceso

3.5. Mecanismos de Extensión

CUMBIA-XLM cuenta con ciertos mecanismos de extensión. Esto es necesario porque CUMBIA-XPM también es extensible. Como se dijo anteriormente, extendiendo elementos de CUMBIA-XPM se pueden soportar lenguajes como BPEL4WS, XPDL, y jBPM.

Así, CUMBIA-XLM debe ser capaz de definir lenguajes que asignan representaciones a elementos extendidos de CUMBIA-XPM.

3.5.1. Representaciones con identificadores

Uno de los mecanismos de extensión es que el estilo permite definir diferentes representaciones para un mismo concepto de CUMBIA-XPM. Esto se hace mediante el atributo *id* del elemento *representation* y referenciando estos identificadores en la decoración. Por ejemplo, en BPEL hay un elemento llamado while, que en Cumbia puede implementarse extendiendo una actividad. Si se quiere definir el lenguaje BPEL en Cumbia se tendría que definir una representación para el while en el estilo. Dicha representación debe ser para una actividad y debe tener un *id* que diferencie esta representación de las demás representaciones que pueden existir para el concepto actividad de CUMBIA-XPM. De esta forma la representación para el while sería algo similar a lo que se muestra en la siguiente figura.

```
<representation concept="activity" id="while">
   . . .
</representation>
```

Figura No. 35: Atributo id de una representación

Suponiendo que se tiene el proceso en BPEL de la Figura No. 37. La decoración de este proceso para Cumbia debe indicar el identificador de la representación para el *while*, este identificador debe coincidir con el identificador (*id*) de la representación de la Figura No. 35. De esta forma se sabe cual es la representación del *while* en el estilo. La Figura No. 36, muestra la decoración de la actividad Solicitud del proceso BPEL, la cual es el *while*. Note que esta decoración tiene un atributo llamado *representationId* y que el valor de éste es igual al identificador de la representación del while en el estilo (atributo *id* del tag *representation*).

```
<activity name="Solicitud" representationId="while">
   . . .
</activity>
```

Figura No. 36: Decoración de una actividad

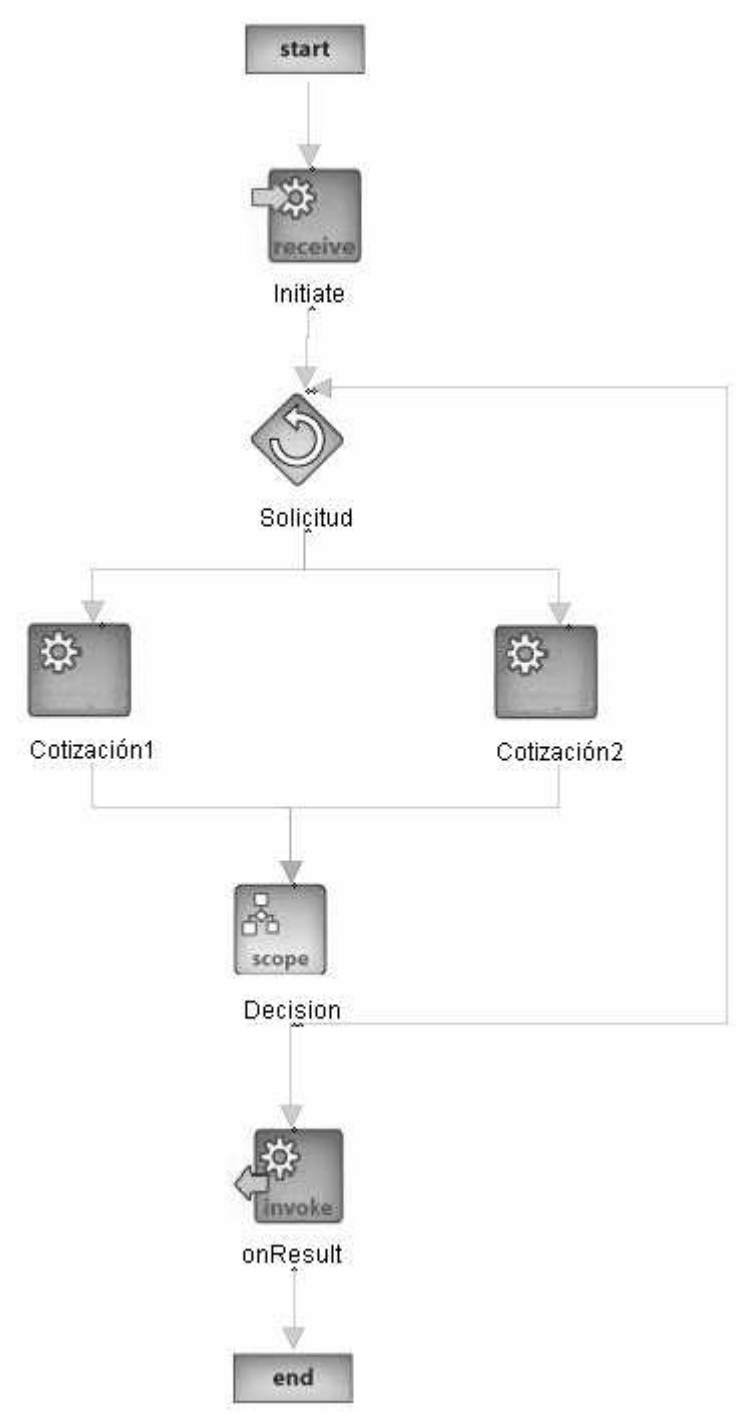

Figura No. 37: Proceso BPEL

Del ejemplo anterior se puede ver que la decoración depende del proceso y del estilo. Ya que en la decoración se especifican los elementos que tiene la definición del proceso y se hace referencia a identificadores de representaciones definidas en el estilo.

3.5.2. Elementos de decoración

Otro mecanismo de extensión, que sirve para enriquecer los lenguajes que se definen con CUMBIA-XLM, es la capacidad de crear representaciones de elementos que no existen en CUMBIA-XPM. Estos elementos no existen en CUMBIA-XPM porque no son de control para la ejecución del proceso, sino que son elementos que dan información al usuario que visualiza el proceso. A estos elementos se les llama **elementos de decoración.**

Los elementos de decoración se definen en la decoración del proceso y deben tener una representación asociada en el estilo.

Un ejemplo claro de un elemento de decoración se presenta en SPEM. Este metamodelo tiene un elemento llamado guía (*guidance*). Una guía no brinda información de control necesaria para ejecutar un proceso, sólo brinda información al usuario que visualiza el proceso.

La notación grafica de una guía según la especificación de SPEM [SPEM05] se muestra en la siguiente figura.

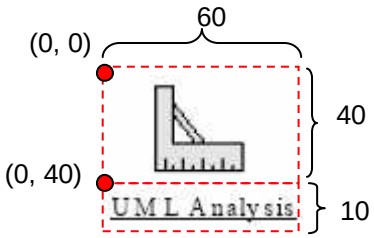

Figura No. 38: Elemento guía de SPEM

La parte del estilo que define la guía se muestra en la Figura No. 39 y un ejemplo de la parte de la decoración que tiene una guía está en la Figura No. 40.

```
<representation id="guia">
   <layout>
      <dimension width="60" height="50"/>
      <position-zone>
         <position x="0" y="0"/>
         <zone>
            <dimension width="60" height="40"/>
            <form>
               <image file="guia.gif"/>
            </form>
            <attributes2 halign="center" valign="center"
                         margin-top="0" margin-bottom="0"
                         margin-right="0" margin-left="0"/>
         </zone>
      </position-zone>
      <position-zone>
         <position x="0" y="40"/>
         <zone>
            <dimension width="60" height="10"/>
            <textfield id="text"/>
            <attributes2 halign="center" valign="center"
                         margin-top="0" margin-bottom="0"
                         margin-right="0" margin-left="0"/>
         </zone>
      </position-zone>
   </layout>
</representation>
```

Figura No. 39: Representación del elemento guidance de SPEM

```
<element name="guia uml" representationId="guia"
         x="5" y="10" width="" height="">
   <text id="text">UML Analysis</text>
</element>
```

Figura No. 40: Guidance como elemento de decoración

3.5.3. Asociaciones

Una asociación no hace parte del modelo CUMBIA-XPM. Sin embargo, CUMBIA-XPM soporta asociaciones entre elementos de un lenguaje. Estas asociaciones son elementos de decoración.

Una asociación se define en el estilo de forma similar a un dataflow. Existe un tipo de representación en el estilo para las asociaciones. La representación de una asociación debe tener un identificador, al cual se debe hacer referencia en la decoración para crear asociaciones en un proceso.

El siguiente es un ejemplo de la representación de una asociación en el estilo.

```
<association-representation id="a1"
                            showName="false"
                            line-pattern="dashed"
                            line-style="squared"
                            line-width="1"
                            begin-direction-type="none"
                            end-direction-type="none" >
   <color r="0" g="0" b="0" alfa="200"/>
</association-representation>
```

Figura No. 41: Representación de una asociación

Esta representación posee el atributo *id*, el cual le da un identificador a la representación. El atributo *showName* indica si se debe mostrar o no el nombre de la asociación. Los demás atributos son los mismos que tiene la representación de un *dataflow*.

El siguiente es un ejemplo de una asociación definida en la decoración.

```
<association name="depends" representationId="a1"
             from="source" to="target">
   <point x="378" y="19"/>
   <point x="478" y="110"/>
</association>
```

Figura No. 42: Decoración de una asociación

Una asociación tiene un origen (atributo *from*) y destino (atributo *to*), en los cuales deben ir los nombres de los dos elementos que están conectados a la asociación.

Se pueden crear asociaciones entre actividades, multiactividades, procesos, y elementos extendidos.

4. Evaluación de CUMBIA-XLM

En este capítulo se presenta una evaluación de la aplicabilidad de CUMBIA-XLM, a partir de la definición de estilos para algunos lenguajes y modelos.

4.1. Definición de Estilos

Para la evaluación de CUMBIA-XLM, se crearon estilos para los principales lenguajes o modelos.

Algunos de estos lenguajes o modelos definen su propia notación grafica, sin embargo algunos no lo hacen. Por esta razón, para los lenguajes o modelos que no tienen una notación grafica definida, se tomó la notación gráfica que utiliza la herramienta de edición más popular para dicho lenguaje.

Una vez creado el lenguaje para Cumbia con CUMBIA-XLM, se crearon procesos de Cumbia para cada lenguaje con su respectiva decoración y se visualizaron con el visualizador de procesos de Cumbia.

4.1.1. BPEL

Para definir el lenguaje BPEL para Cumbia se tomó la notación gráfica que utiliza la herramienta de edición Oracle BPEL PM Designer. La siguiente figura muestra un ejemplo de un proceso en esta herramienta.

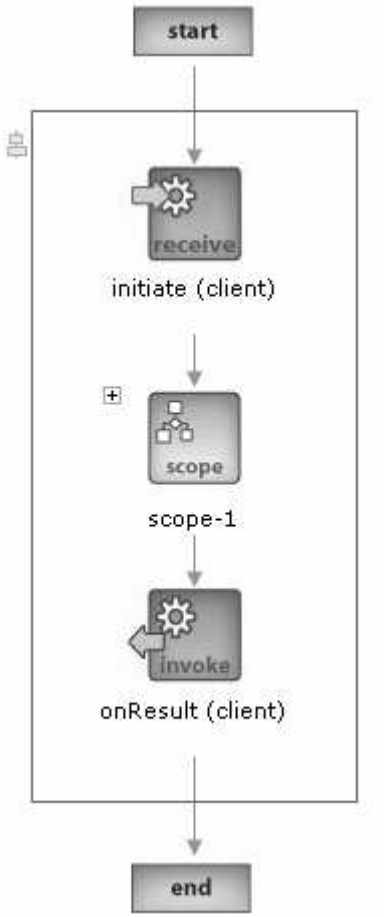

Figura No. 43: Ejemplo de un proceso en BPEL

Como se puede ver, BPEL no tiene el concepto de puertos, por tal razón para los puertos de entrada y salida se asignan representaciones no visibles en el estilo, es decir se crearon representaciones para el puerto de entrada y salida como un punto que no es visible.

BPEL tiene un elemento llamado *scope* (ver Figura No. 43), el cual define un espacio en el que se pueden agrupar otros elementos. Este elemento corresponde a un proceso en Cumbia, de esta forma la representación del *scope* de BPEL es la representación de un proceso en Cumbia. Así, se creó una representación en el estilo para el proceso, de tal forma que fuera la misma al grafismo utilizado para el *scope*. En la Figura No. 44 se muestra la distribución de las zonas de la representación del *scope*.

Otros elementos de BPEL son: *invoke*, *receive*, *empty*, *wait*, *compensate*, *throw*, *while*, *switch*, *flow*. Para cada uno de estos elementos se crea una representación en el estilo. Las representaciones de estos elementos se toman como actividades extendidas de Cumbia.

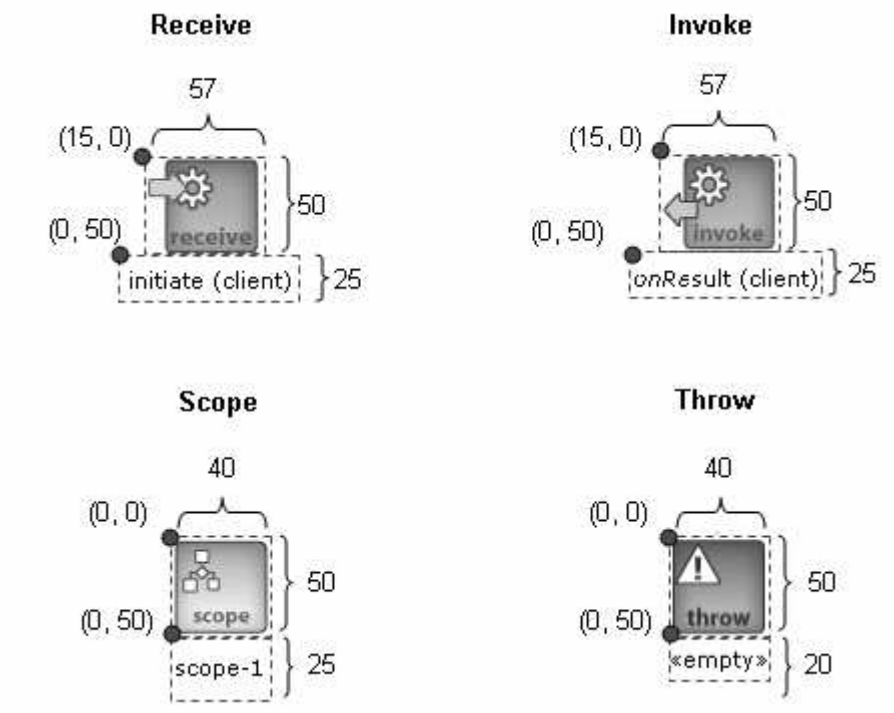

Figura No. 44: Representaciones para elementos de BPEL

El estilo para BPEL tiene dos elementos de decoración los cuales indican el punto de inicio y el punto de finalización del proceso. Estos puntos de inicio y fin de proceso se representan con la siguiente notación en la herramienta Oracle BPEL PM Designer.

Figura No. 45: Representaciones para inicio y fin de un proceso en BPEL

Todas las representaciones para BPEL son sencillas de hacer, ya que cada elemento tiene una imagen definida e inmediatamente debajo de la imagen tiene el nombre que se le asigna al elemento. También, cada representación de estas tiene dos zonas: una para los puertos de entrada y otra para los puertos de salida. En la Figura No. 46 se muestra la distribución de estas zonas.

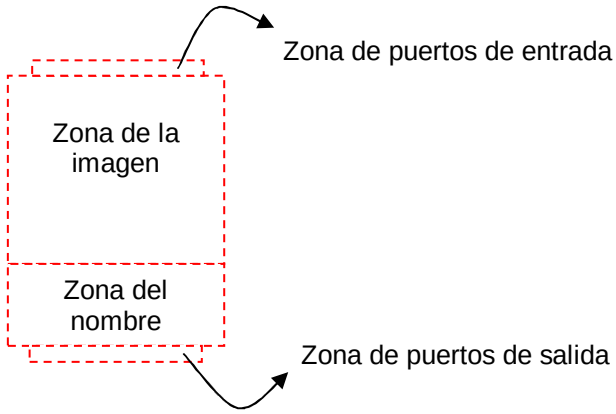

Figura No. 46: Zonas de las representaciones para elementos de BPEL

El siguiente es el proceso Cumbia que se creó utilizando el lenguaje para BPEL definido con CUMBIA-XLM.

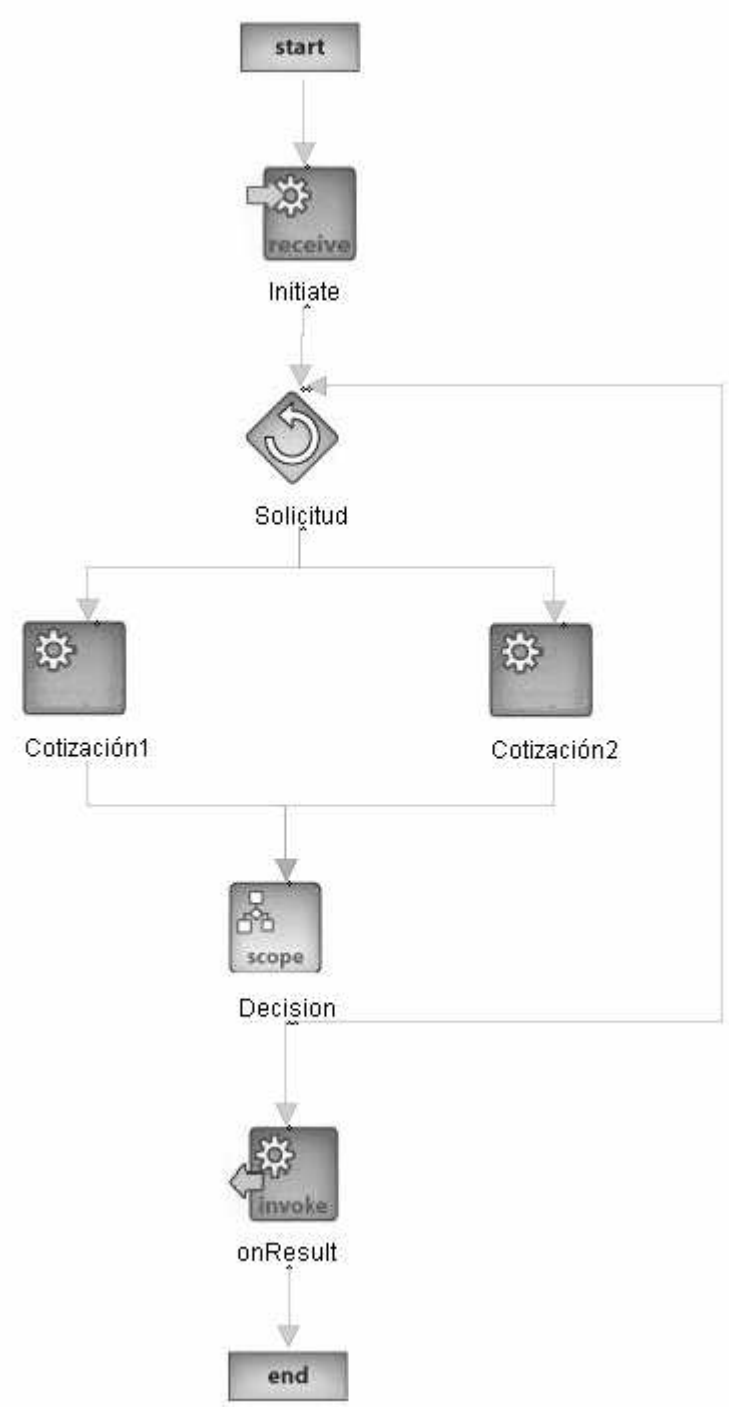

Figura No. 47: Proceso BPEL usando CUMBIA-XLM

4.1.2. XPDL

A pesar de que XPDL no tiene una notación grafica definida para sus elementos, se tomó la notación que utiliza la herramienta de edición JaWE.

La siguiente figura muestra un ejemplo de un proceso XPDL editado en JaWE.

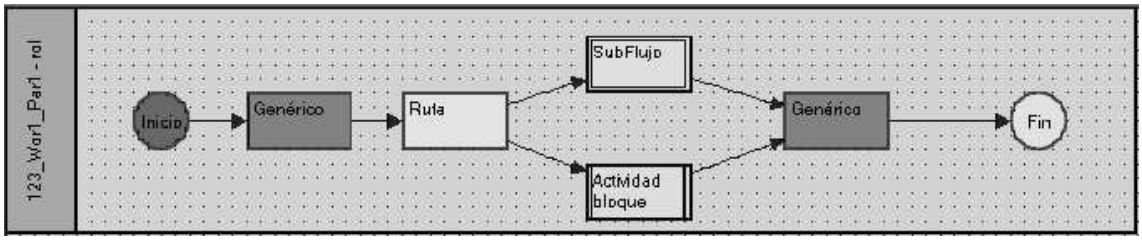

Figura No. 48: Proceso XPDL en JaWE

En XPDL existen tres tipos de actividades: actividades *implementation*, las cuales son procedimientos manuales o automáticos; actividades *route*, las cuales sirven para enrutar el flujo de ejecución; y actividades *block*, las cuales son un conjunto de actividades. Además hay subflujos, los cuales contienen otro proceso. Otro elemento de XPDL es el *swimlane* el cual contiene actividades y subflujos para indicar que dichas actividades y subflujos tienen un participante o responsable.

En el ejemplo anterior, las actividades llamadas “Genérico” son actividades *implementation;* la actividad llamada “Ruta” es una actividad de tipo *route*; y la actividad llamada “Actividad bloque” es una actividad de tipo *block*. El *swimlane* es el área rectangular que abarca todas las actividades y en la parte izquierda tiene el nombre del participante. Además este proceso tiene dos círculos que indican el inicio y fin del proceso.

XPDL no tiene el concepto de puertos, por lo tanto en el estilo se definen representaciones no visibles para los puertos de entrada y salida.

El estilo para XPDL define representaciones para cada uno de los elementos mencionados anteriormente.

Para cada actividad de XPDL se crea una representación con un identificador que indica el tipo de la actividad, como se muestra en la Figura No. 49. La distribución de las zonas para las actividades consiste en una zona para el rectángulo que delimita la actividad, una zona para el nombre de la actividad, y zonas a los lados para puertos de entrada y puertos de salida, tal como se ve en la Figura No. 50.

```
<representation concept="activity" id="implementation">
 . . .
</representation>
<representation concept="activity" id="route">
 . . .
</representation>
<representation concept="activity" id="block">
 . . .
</representation>
```

Figura No. 49: Representaciones para actividades de XPDL

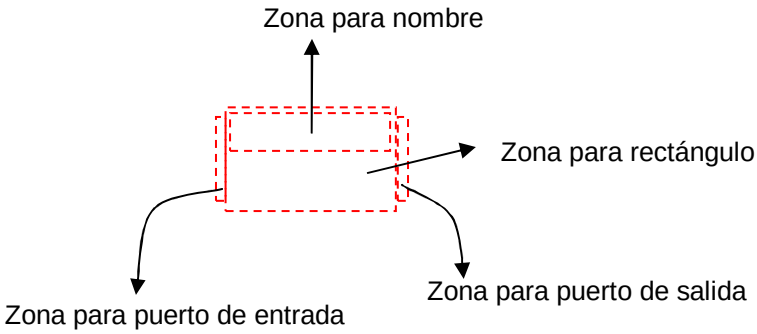

Figura No. 50: Zonas para la representación de una actividad de XPDL

La actividad de tipo bloque no se representa con un rectángulo simple, sino que es un rectángulo que tiene líneas dobles en los lados izquierdo y derecho. Esto se hace en el estilo con dos zonas: una zona que contiene el rectángulo que delimita toda la actividad, y otra zona con un rectángulo más pequeño contenido en el anterior rectángulo de tal forma que se ven las líneas dobles en los costados de la actividad. Las zonas de la representación de la actividad de tipo bloque se muestran en la siguiente figura.

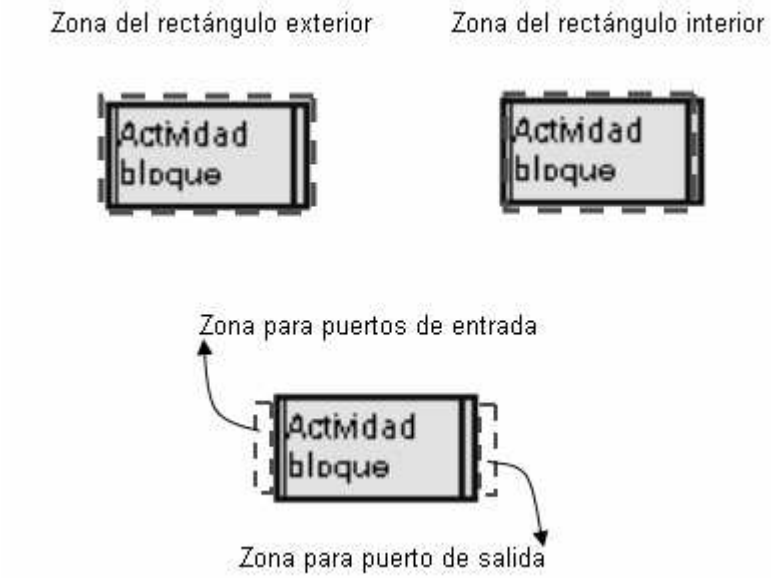

Figura No. 51: Zonas para la representación de la actividad block de XPDL

La representación del subflujo se crea de forma similar a la representación de la actividad *block*, porque el rectángulo de un subflujo tiene líneas dobles en todos los lados.

El otro elemento que se necesita definir en el estilo es *swimlane*. Este elemento es de decoración ya que este elemento no afecta la ejecución del proceso en Cumbia. La representación de este elemento se hace con dos zonas que contienen rectángulos y una zona que contiene el nombre del participante o responsable.

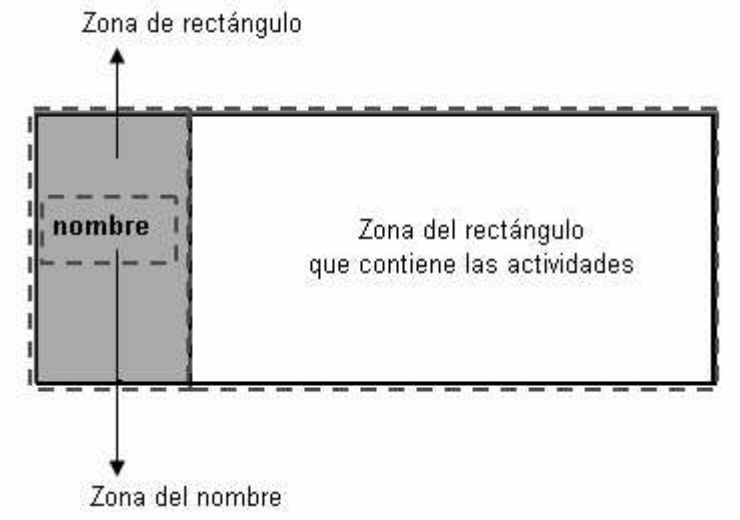

Figura No. 52: Zonas de la representación del swimlane de XPDL

Otros elementos de decoración son los que indican el inicio y fin del proceso. En la siguiente figura se muestran las zonas de estos elementos. Una zona contiene la figura geométrica (circulo), y la otra zona contiene un *label* (“inicio” o “fin”).

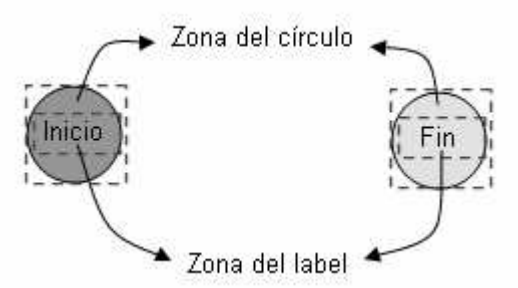

Figura No. 53: Zonas de las representaciones de inicio y fin de proceso XPDL

La siguiente figura muestra un proceso utilizando el estilo definido para XPDL.

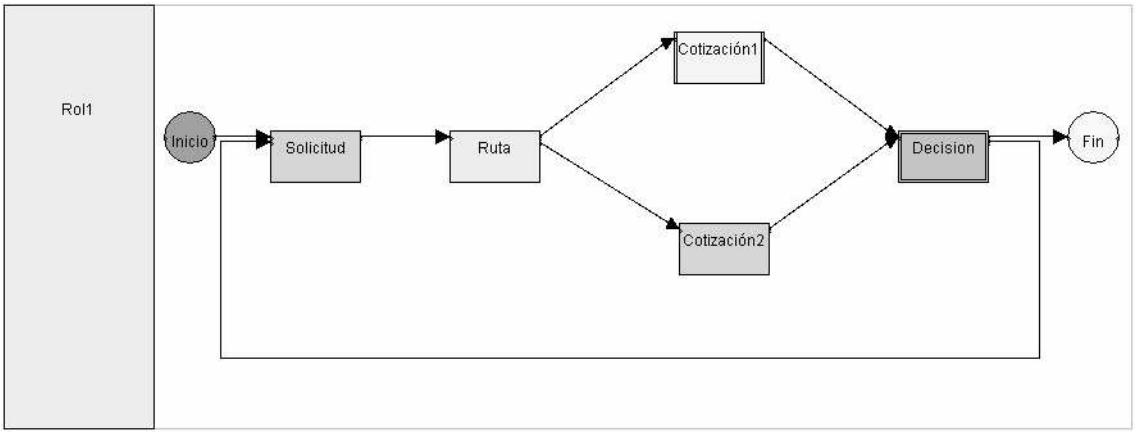

Figura No. 54: Proceso XPDL usando CUMBIA-XLM

4.1.3. IMS-LD

IMS-LD no define notaciones graficas para sus elementos. Por tal razón, para la creación de un estilo para IMS-LD se utilizó la notación de un editor grafico utilizado para editar procesos IMS-LD. Este editor es MOT+.

El siguiente es un ejemplo de un proceso editado con MOT+ (Tomado de [MCVCC05]).

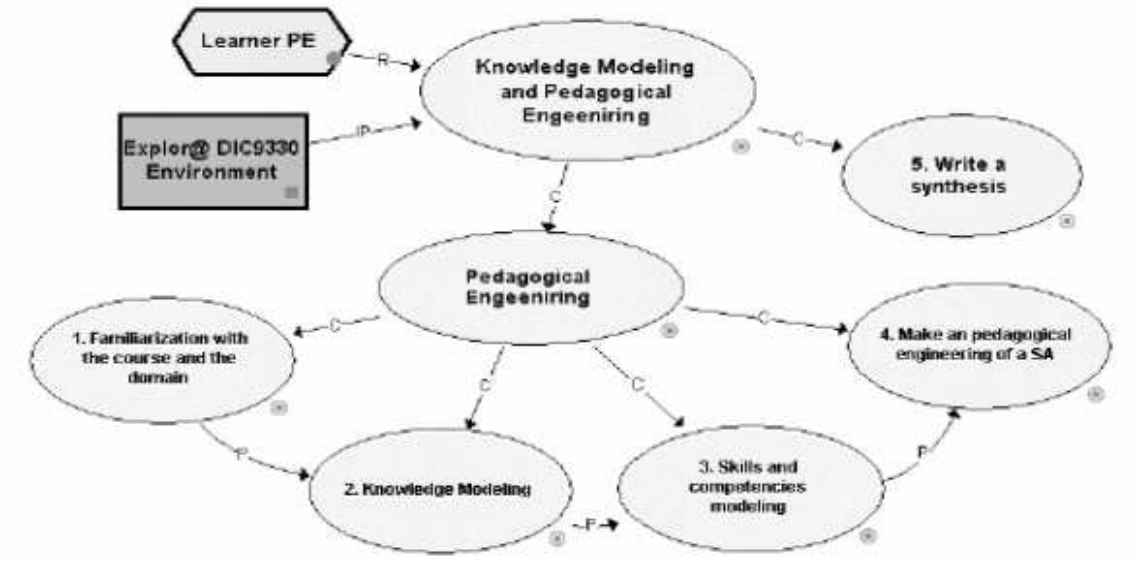

Figura No. 55: Proceso IMS-LD en MOT+

Los elementos que muestra la figura anterior son: actividad, rol, y ambiente (*environment*). Los óvalos son las actividades, el rol es el hexágono con nombre "Learner PE" y el ambiente es el rectángulo.

IMS-LD no tiene el concepto de puerto por lo que se define una representación para puerto de entrada y puerto de salida de tal forma que no sean visibles.

El estilo para IMS-LD define una representación para la actividad que consiste en una zona para el ovalo, una zona para el nombre de la actividad, una zona para los puertos de entrada y otra zona para los puertos de salida.

Para el elemento rol de IMS-LD se creó una representación con una zona que contiene el hexágono como una imagen y otra zona que contiene el nombre del rol. Además se creó una representación de asociación para permitir las asociaciones entre actividades y roles.

Para el elemento *environment* de IMS-LD se creó una representación de elemento de decoración con una zona que contiene un rectángulo y otra zona que contiene el identificador del *environment*.

Las asociaciones entre las actividades son *dataflows* en Cumbia, para estas se creó una representación de *dataflows* que muestran el nombre del *dataflow* mediante el atributo *showName* del elemento *dataflow-representation* del estilo.

La siguiente figura muestra un proceso IMS-LD con CUMBIA-XLM.

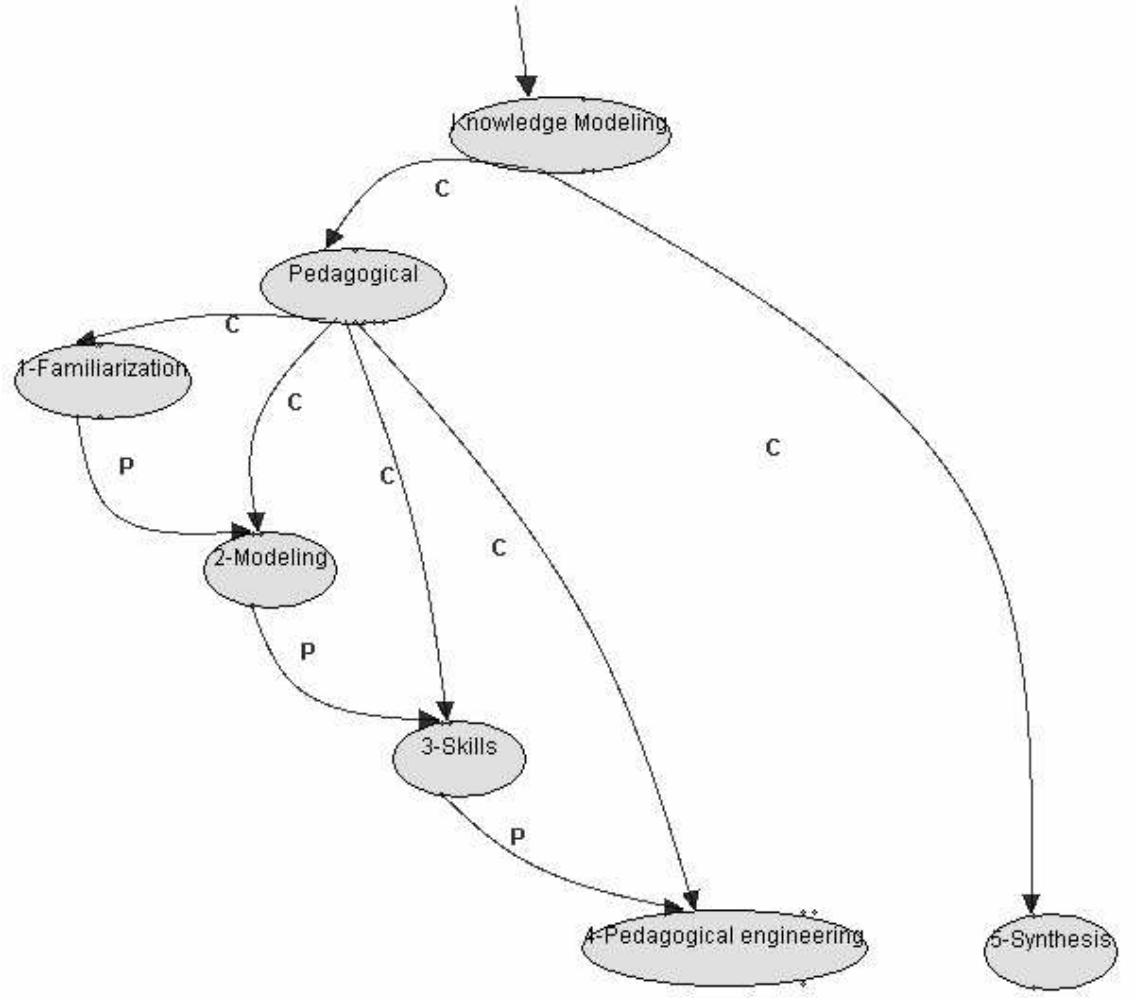

Figura No. 56: Proceso IMS-LD usando CUMBIA-XLM

4.1.4. SPEM

La especificación de SPEM define grafismos a sus elementos. De esta especificación [SPEM05] se tomó un ejemplo de un proceso en SPEM, el cual se muestra en la siguiente figura.

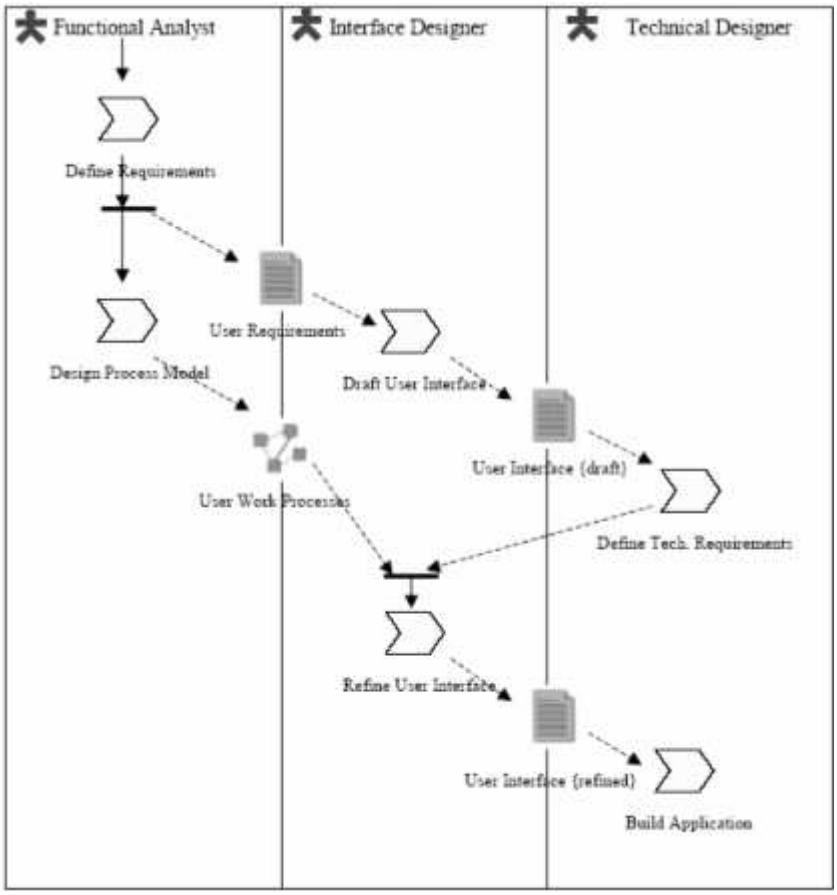

Figura No. 57: Proceso en SPEM

Los elementos que se pueden ver en el anterior proceso son: actor, actividad, documento, modelo UML. Además, se puede dividir el diagrama en partes para cada actor, indicando las actividades que realiza cada actor. Por ejemplo, las actividades "Define Requeriments" y "Design Process Model" son realizadas por el actor "Functional Analyst".

Las representaciones del estilo para SPEM son sencillas, y consisten de una zona que tiene una imagen con el grafismo definido para el elemento, otra zona para el nombre del elemento y dos zonas a los lados para los puertos de entrada y salida.

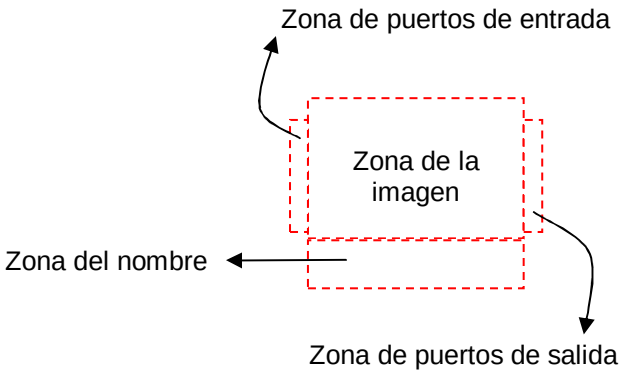

Figura No. 58: Zonas de las representaciones para elementos de SPEM

La siguiente figura muestra las representaciones de algunos elementos de SPEM, con la distribución de las zonas tal como se explico anteriormente. Estos elementos se representaron como actividades de Cumbia.

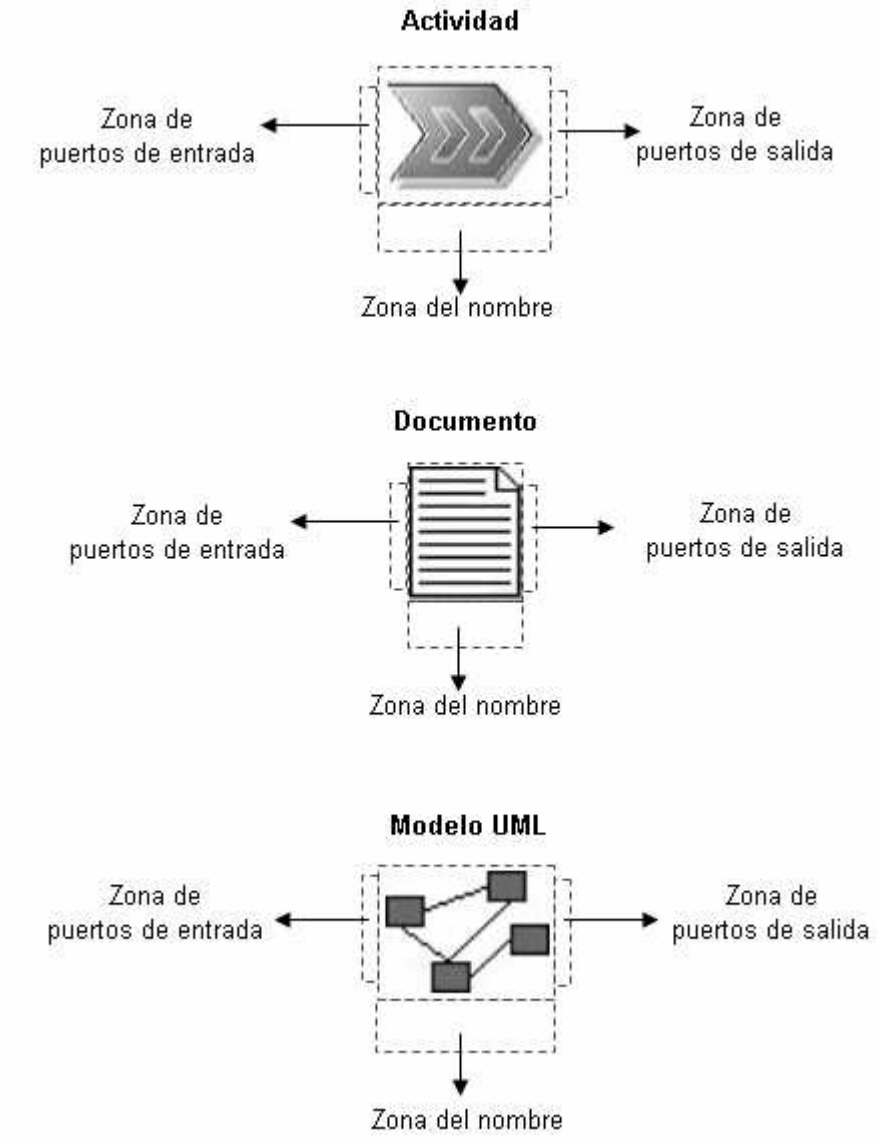

Figura No. 59: Representaciones de elementos para SPEM

Para los puertos de entrada y salida se crearon representaciones no visibles que consisten en una zona de ancho y alto igual a uno, la cual contiene un ovalo. Esto hace que los puertos no sean visibles en los procesos SPEM que se creen utilizando el lenguaje para SPEM definido por CUMBIA-XLM.

SPEM tiene diferentes tipos de asociaciones entre sus elementos. Para cada tipo de asociación se creó una representación. Por ejemplo, para las asociaciones de SPEM que se ven entre las actividades del proceso de la Figura No. 57, se creó una representación del *dataflow* con una línea sólida, con una flecha al final, y sin nombre. Otro tipo de asociaciones que se ven en esta figura, son las líneas punteadas, estas se representaron también como *dataflows*, es decir, que se creó otra representación para el *dataflow* con la línea punteada.

SPEM tiene un elemento llamado rol, este elemento se representa con el grafismo de actor de UML. CUMBIA-XLM permite definir representaciones para recursos. Estos recursos pueden ser roles. Por tal razón el rol de SPEM se toma como una representación de un recurso en el estilo. La siguiente figura muestra la representación del rol de SPEM como un recurso de Cumbia.

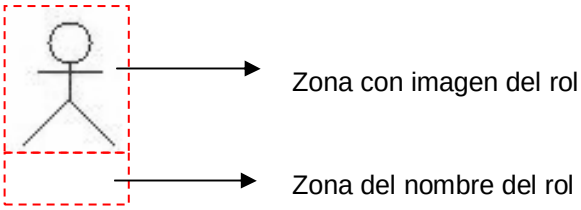

Figura No. 60: Representación del elemento rol de SPEM

Además se creó una representación de un elemento de decoración que corresponde a las divisiones del proceso que se ven en la Figura No. 57. Esta representación sólo consiste de una zona con un rectángulo.

Otro elemento que tiene SPEM es la guía (*guidance*). Este elemento sirve para hacer comentarios o notas aclaratorias sobre elementos en el proceso. La Figura No. 39 muestra esta representación.

Debido a que la guía se puede asociar a otros elementos de SPEM mediante una asociación, se creó una representación de una asociación con una línea punteada y sin dirección.

La siguiente figura muestra el proceso de la Figura No. 57, pero utilizando CUMBIA-XLM.

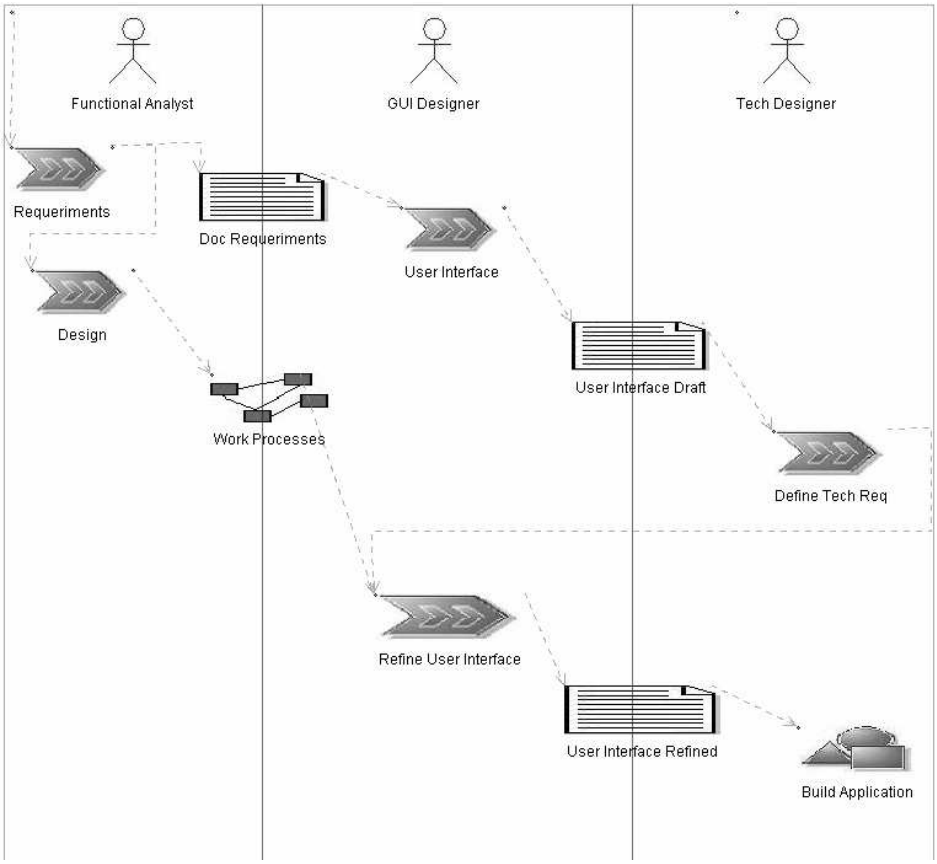

Figura No. 61: Proceso en SPEM usando CUMBIA-XLM

4.1.5. BPMN

BPMN define una notación grafica para los elementos necesarios en la modelación de procesos de negocio. De [Whi04] se tomó la siguiente figura, la cual es un ejemplo de un proceso en BPMN.

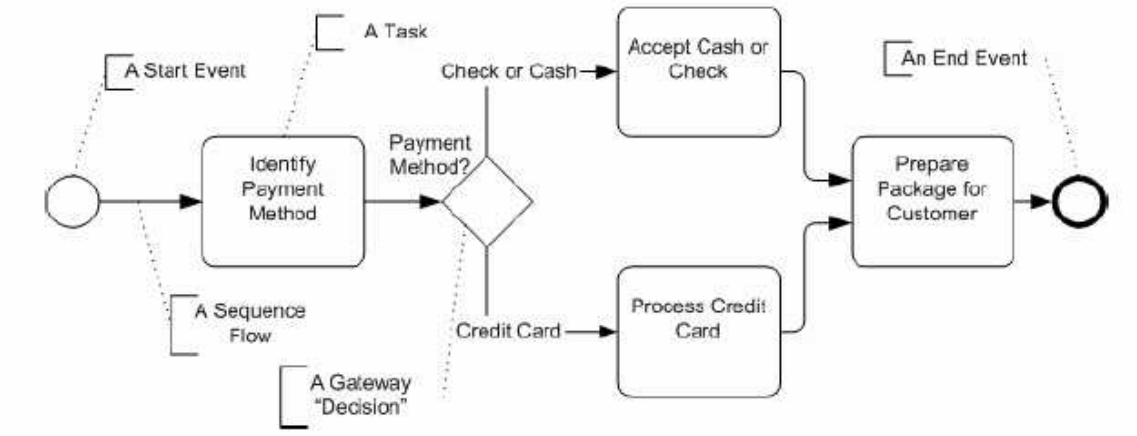

Figura No. 62: Proceso en BPMN

El círculo inicial es un evento inicial y el círculo final es un evento final. Los rectángulos redondeados son actividades, y el rombo es un elemento llamado *gateway*. Además

hay comentarios asociados a cada elemento, y las asociaciones del *gateway* tienen nombre.

BPMN no tiene el concepto de puertos, por lo tanto se crearon representaciones para el puerto de entrada y de salida de tal forma que no se ven.

Además del evento inicial y final existe el evento intermedio. BPMN también tiene el concepto de proceso.

Los eventos, la actividad y el *gateway* son considerados actividades de Cumbia. Y el proceso de BPMN es directamente representado con el proceso de Cumbia. La siguiente figura muestra la distribución de las zonas para las representaciones de los eventos.

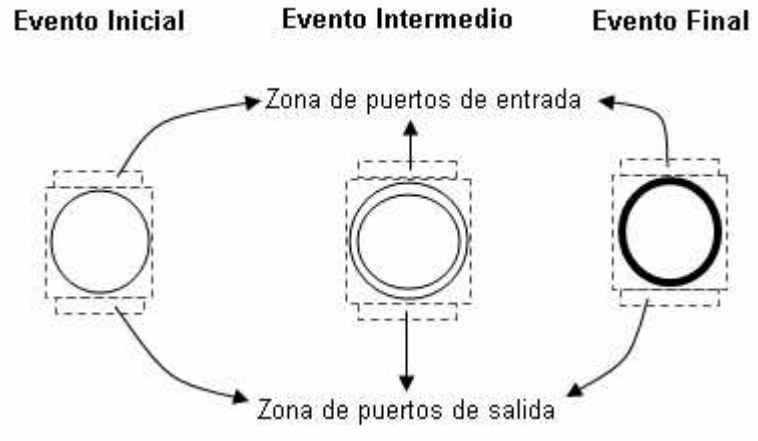

Figura No. 63: Representaciones para eventos de BPMN

La distribución de las zonas de las representaciones de la actividad, el proceso y el *gateway* se muestra en la siguiente figura.

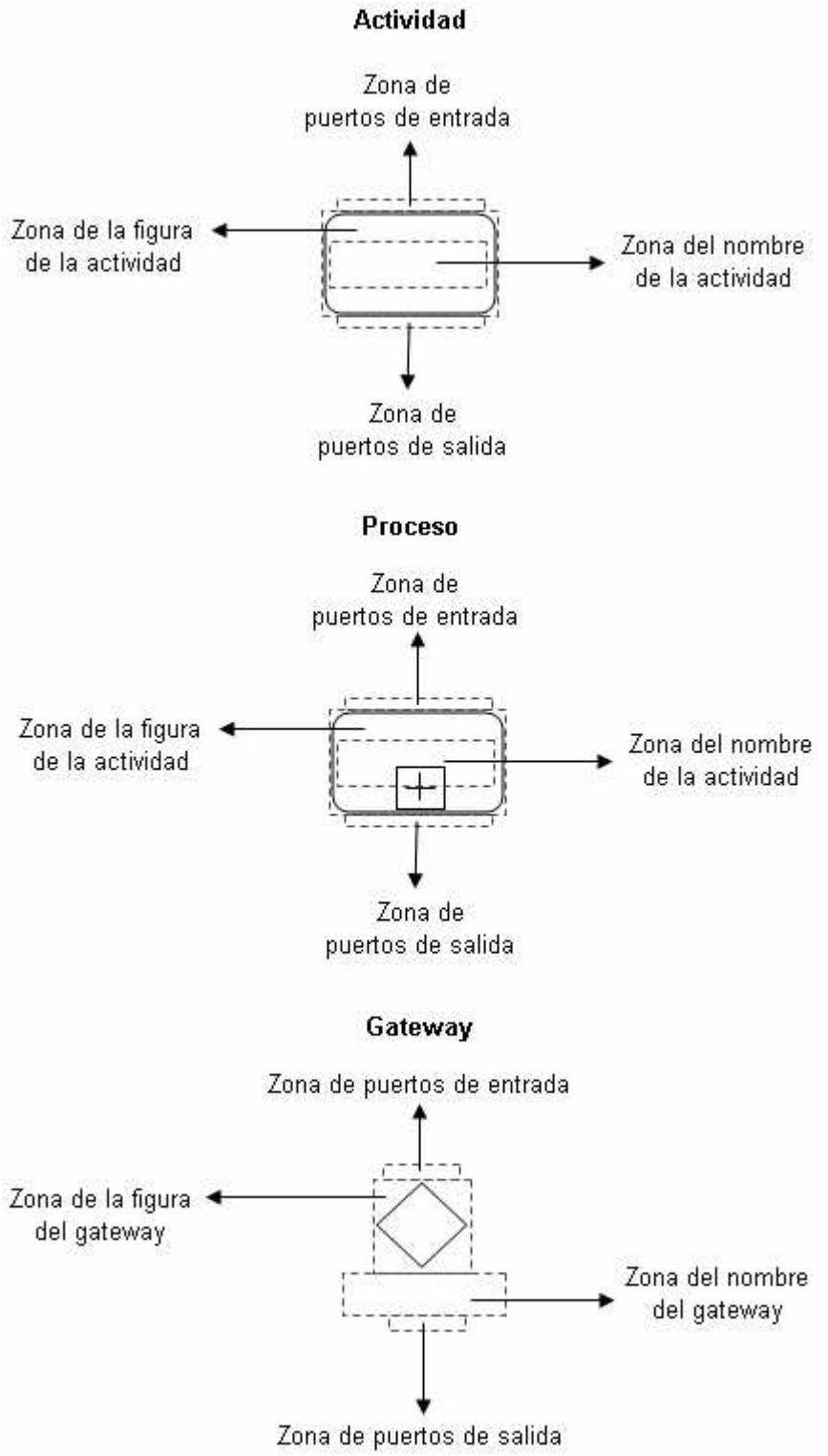

Figura No. 64: Representaciones de BPMN

Se crearon dos representaciones para el *dataflow*, una que no muestra el nombre de la asociación, y otra que si muestra el nombre de la asociación para el caso del *gateway*. Además, se creó una representación para la asociación con los comentarios, la cual es una línea punteada.

Otros elementos que posee BPMN, son *pool* y *lane*. Un pool representa un participante en el proceso, y contiene actividades, procesos, *gateways* y eventos. Un *lane* divide un pool en partes para categorizar las actividades, procesos, *gateways* y eventos del pool. Para estos elementos se crearon representaciones de elementos de decoración, tal como se muestra en la siguiente figura.

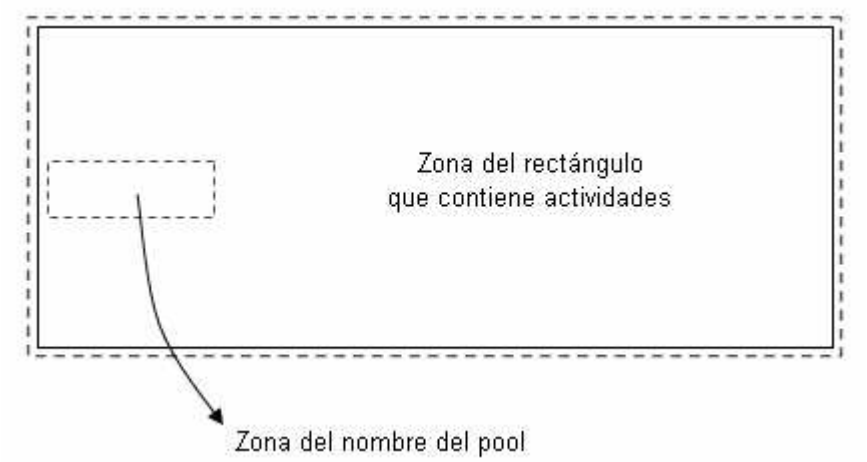

Figura No. 65: Representación del pool de BPMN

La representación del *lane* es igual a la representación del *pool*.

Finalmente, la siguiente figura es un proceso BPMN utilizando CUMBIA-XLM.

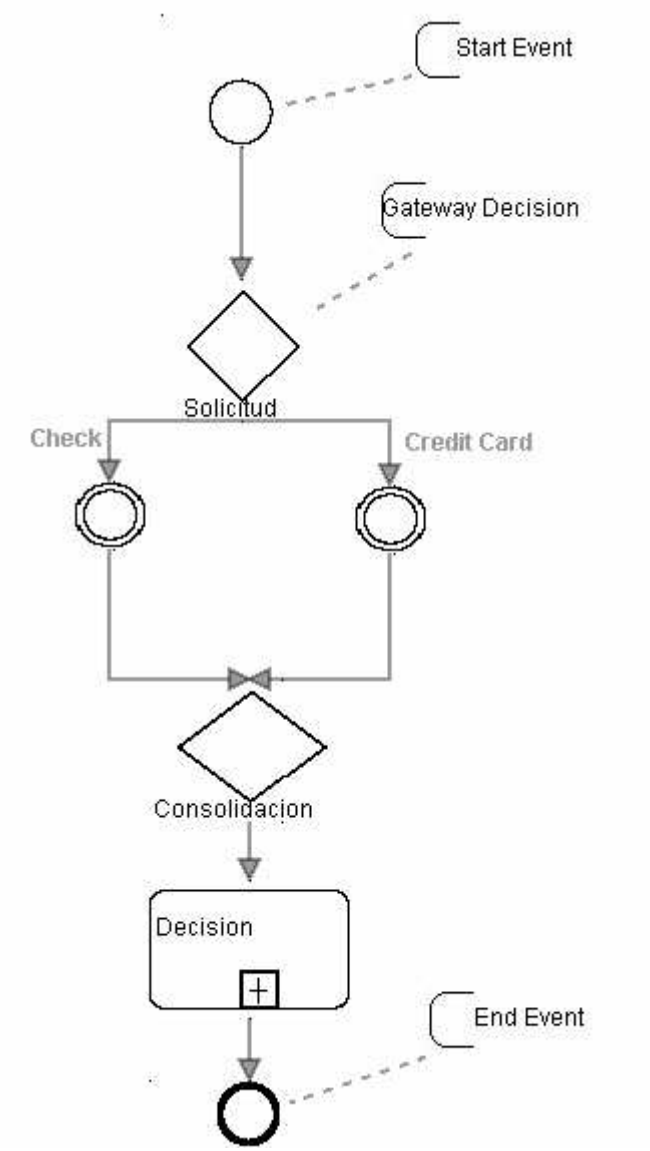

Figura No. 66: Proceso BPMN usando CUMBIA-XLM

4.1.6. WSFL

La especificación de WSFL [Ley01] no define una notación grafica estricta. Sin embargo, de esta especificación se tomó un ejemplo de un proceso, el cual se muestra en la siguiente figura.

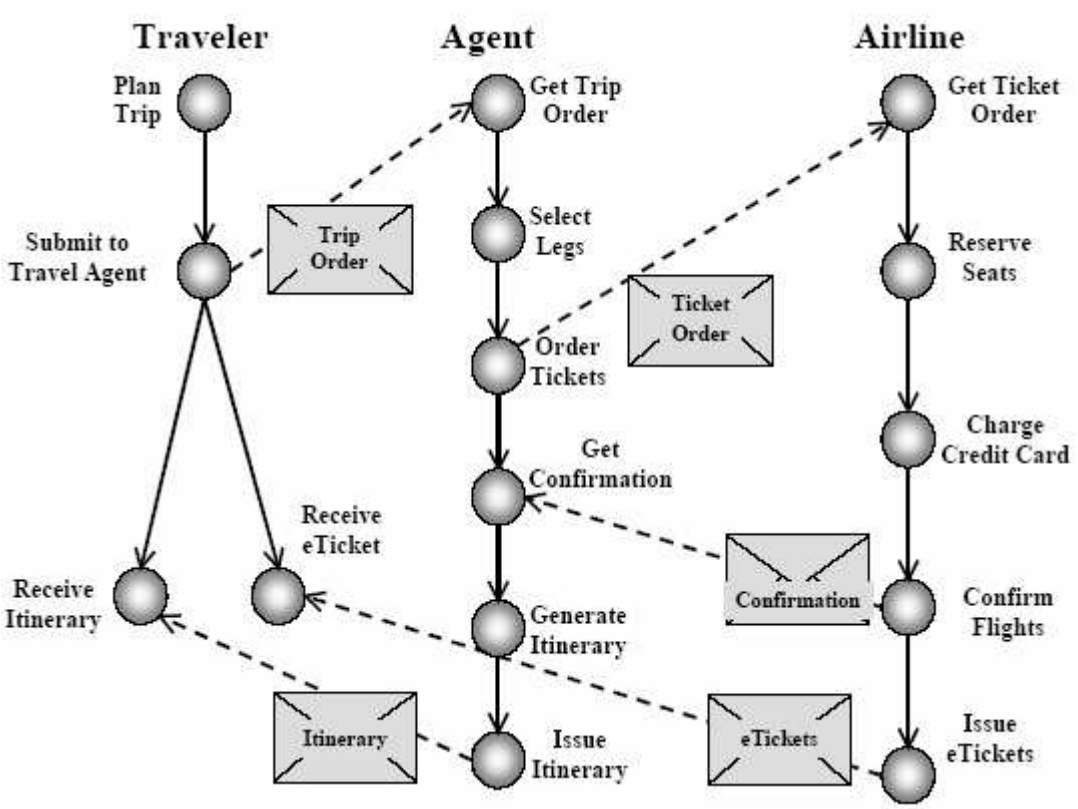

Figura No. 67: Proceso en WSFL

De acuerdo al proceso anterior, WSFL tiene actividades, las cuales son los círculos grises, y tiene dos tipos de enlaces o asociaciones: las líneas sólidas que indican flujo de control y las líneas punteadas que indican flujo de datos o mensajes. Además, en este proceso se ven un elemento similar a un sobre de carta el cual aparece sobre los flujos de mensajes, indicando los datos que viajan por estas asociaciones.

WSFL no tiene el concepto de puertos, por lo tanto no se necesita una representación visible de los puertos, es decir que en el estilo se crean representaciones con una zona de ancho y alto igual a 1, de tal forma que no se vean los puertos de entrada y de salida en el proceso.

Otro elemento que se ve en el proceso anterior es el rol. En la parte superior de la figura aparecen los nombres de los roles (Traveler, Agent, Airline) que tienen asignadas las actividades.

La representación de la actividad tiene una zona con la imagen del círculo gris, otra zona con el nombre de la actividad, y dos zonas para los puertos de entrada y salida.

La representación del dato tiene una zona con la imagen del sobre de carta y otra zona con el nombre del dato.

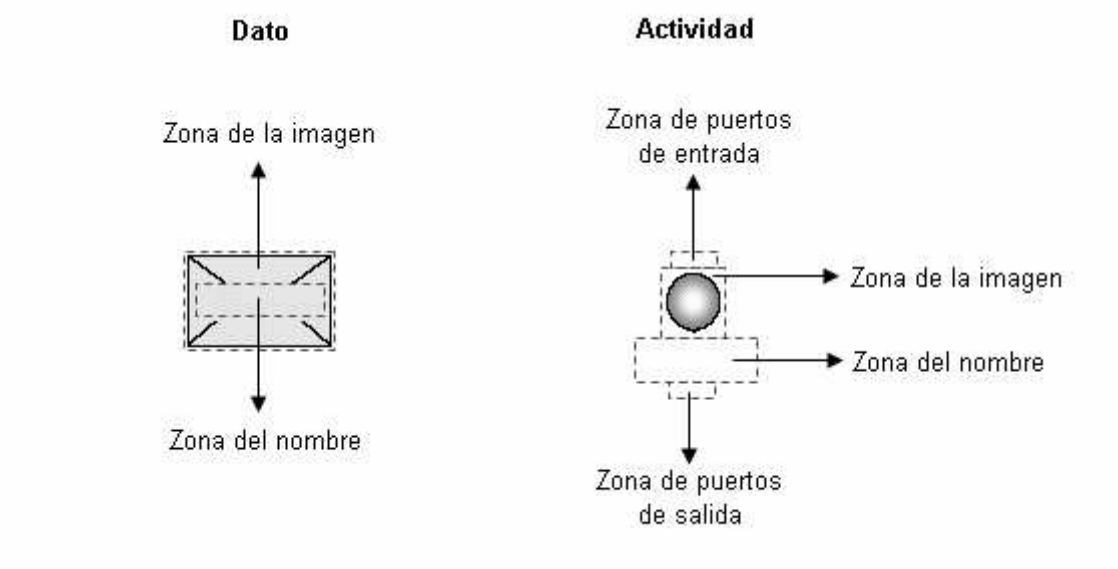

Figura No. 68: Representaciones de WSFL

Otra representación que se creó en el estilo es la del recurso. En este caso la representación del recurso consiste de una sola zona que contiene el nombre del recurso.

Además, se crearon dos representaciones para el *dataflow*, una representación con una línea sólida para los flujos de control y otra representación con una línea punteada para los flujos de datos.

La siguiente figura muestra un proceso WSFL utilizando CUMBIA-XLM.

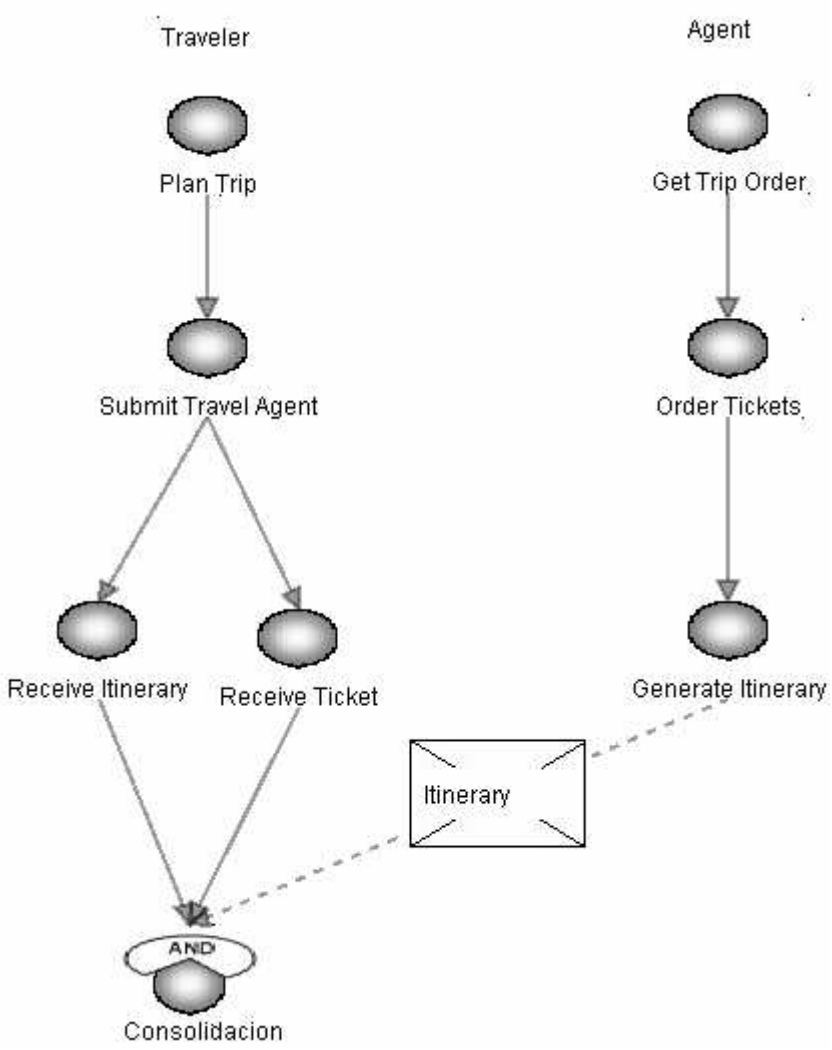

Figura No. 69: Proceso WSFL usando CUMBIA-XLM

4.2. Balance

De acuerdo a la tarea de construir los estilos de CUMBIA-XLM que se explicaron en la sección anterior, se puede concluir que CUMBIA-XLM es aplicable a los principales lenguajes y modelos de definición de procesos.

Algunos estilos fueron más complicados de hacer que otros debido a la complejidad y cantidad de los elementos que poseen.

Los atributos de la representación para los *dataflows* en el estilo fueron suficientes para crear las representaciones de los *dataflows* que se necesitaron. Además, estas representaciones fueron sencillas de hacer en los estilos. Estos atributos sirven para especificar las características que deben tener los *dataflows*; características como: el tipo de línea del *dataflow*, el color de la línea, el tipo de decoración al inicio y fin de la línea, y si se muestra o no el nombre del *dataflow*.

Ninguno de los lenguajes y modelos que se tuvieron en cuenta tienen el concepto de puerto que maneja Cumbia. Por tal razón, en todos los estilos se llevó a cabo la misma estrategia: crear representaciones para el puerto de entrada y el puerto de salida de tal

forma que no sean visibles. Esta es la estrategia que se debe tomar cuando se quiere crear un estilo para un lenguaje que no tiene puertos.

La mayoría de las representaciones que se crearon en los estilos utilizaron imágenes para el grafismo de cada elemento del lenguaje. Esta forma de crear representaciones es bastante sencilla.

Sin embargo, crear representaciones que utilizan figuras geométricas es una tarea con un grado mayor de complejidad, aunque sigue siendo una tarea posible y viable. La dificultad de crear las representaciones usando figuras geométricas radica en que para construir figuras complejas con las figuras geométricas primitivas (rectángulo, rectángulo redondeado, y óvalo) que ofrece el estilo hay que definir diferentes zonas y en cada una de ellas especificar que figura geométrica debe ir. Es muy posible que estas zonas se sobrepongan, por lo que hay que tener cuidado en el orden en que se crean las zonas en el estilo, ya que este orden es el que define el orden en que se sobreponen las zonas. Además, para cada figura geométrica se debe especificar el color, si la figura tiene borde y si tiene relleno.

Una recomendación que se obtiene de este análisis es que para representaciones con grafismos complejos es mejor usar imágenes y para representaciones con grafismos sencillos es mejor usar figuras geométricas para que los estilos no queden demasiado cargados de imágenes que pueden disminuir la eficiencia a la hora de visualizar los procesos.

Las limitaciones que se encontraron son:

- Las representaciones para la actividad, multiactividad y proceso tienen zonas fijas para los puertos de entrada y los puertos de salida, lo cual hace que la ubicación de los puertos sea algo rígida. Aunque, esto también puede ser visto como una ventaja, ya que si los puertos de entrada y salida están ubicados en zonas fijas facilita la identificación de los mismos en la visualización de un proceso.
- No existen opciones en el estilo para definir ciertos atributos del texto, como el color, la fuente, el tamaño de la fuente, y si el texto debe aparecer horizontal o vertical. En el estilo la información textual tiene la misma fuente, el mismo tamaño y aparece en forma horizontal.

De la experiencia de crear los estilos, se puede decir que lo difícil no es crear los estilos: lo difícil es decidir cómo manejar elementos que no existen en Cumbia. Es decir, la creación de los estilos no consistió solamente en una asignación de grafismos a los elementos de Cumbia, sino que consistió en decidir cómo representar elementos como el *while* de BPEL o el *guidance* de SPEM en Cumbia.

En general, la tarea de crear estos estilos muestra que CUMBIA-XLM brinda las herramientas necesarias para definir lenguajes reales.

En la siguiente figura se muestra un cuadro que ilustra la correspondencia de los elementos de Cumbia con BPEL, XPDL, IMS-LD, SPEM, BPMN y WSFL. En este cuadro se puede ver que hay elementos de Cumbia que no tienen un elemento correspondiente en los lenguajes y modelos estudiados, tales como el puerto y la multiactividad. Además, Los elementos de los lenguajes y modelos que no tienen una correspondencia directa se pueden expresar con otros elementos de Cumbia.

	BPEL	XPDL	IMS-LD	SPEM	BPMN	WSFL
Puerto	Representaciones no visibles	Representaciones no visibles	Representaciones no visibles	Representaciones no visibles	Representaciones no visibles	Representaciones no visibles
Workspace	Sin representación	Sin representación	Sin representación	Sin representación	Sin representación	Sin representación
Actividad	Representaciones para elementos: *invoke*, *receive*, *empty*, *wait*, *compensate*, *throw*, *while*, *switch*, *flow*	Representaciones para las actividades: *implementation, route y block*	Representación para actividad	Representaciones para actividad, documento, modelo UML	Representaciones para actividad, eventos y *gateway*	Representación para actividad
Mutliactividad	Sin representación	Sin representación	Sin representación	Sin representación	Sin representación	Sin representación
Proceso	Representación para *scope*	Representación para subflujo	Sin representación	Representación para proceso	Representación para proceso	Sin representación
Dataflow	Una sola representación	Una sola representación	Una sola representación	Diferentes representaciones para cada una de las asociaciones	Dos representaciones: una q muestra el nombre del dataflow y otra que no	Dos representaciones: una con línea solida y otra con línea punteada
Elementos de decoración	Representaciones para indicar el inicio y fin del proceso	Representaciones para *swimlane* y para el inicio y fin del proceso	Representaciones para *environment*	Representaciones para *guidance*, asociaciones entre *guidance* y *swimlane*	Representaciones para *pool* y *lane*	Representación para dato

5. Arquitectura

En este capítulo se expone la arquitectura de la implementación de Cumbia y se explican cada una de sus partes, enfocándose en las partes que involucran a CUMBIA-XLM. La primera sección explica la arquitectura y la segunda muestra como interactúan los componentes de esta arquitectura durante la ejecución de un proceso.

5.1. Arquitectura Global

Los usuarios de Cumbia pueden ser de diferente tipo. Los tipos de usuarios de Cumbia son: desarrollador de motores, administrador, desarrollador de aplicaciones, y cliente.

El desarrollador de motores se encarga de extender CUMBIA-XPM mediante los mecanismos de extensión y de crear lenguajes mediante los estilos de CUMBIA-XLM.

El administrador es el usuario encargado de las tareas administrativas sobre los motores de Cumbia, tales como monitorear el estado del los motores de Cumbia, y modificar propiedades de configuración de los motores de Cumbia.

El desarrollador de aplicaciones crea los procesos, asigna el estilo para el proceso, crea la decoración del estilo, define las políticas de asignación de recursos, y crea o proporciona los recursos del proceso.

El cliente es el usuario de las aplicaciones que se ejecutan sobre Cumbia. Este tipo de usuario puede iniciar la ejecución de un proceso, llevar a cabo actividades de un proceso, visualizar un proceso en ejecución, crear, modificar y eliminar datos, y asignar recursos.

Cada uno de los tipos de usuarios tiene un conjunto de herramientas de software que facilitan la realización de las funciones que tiene asignadas. Una de las herramientas usadas por el desarrollador de motores es el editor de estilos. Como el desarrollador de motores necesita extender los elementos de CUMBIA-XPM, se necesitan editores como Eclipse para crear las clases en Java que extienden de los elementos de CUMBIA-XPM.

El administrador necesita de una consola de administración. El desarrollador de aplicaciones necesita de un editor de aplicaciones. El cliente hace uso de una aplicación disponible a través del Web.

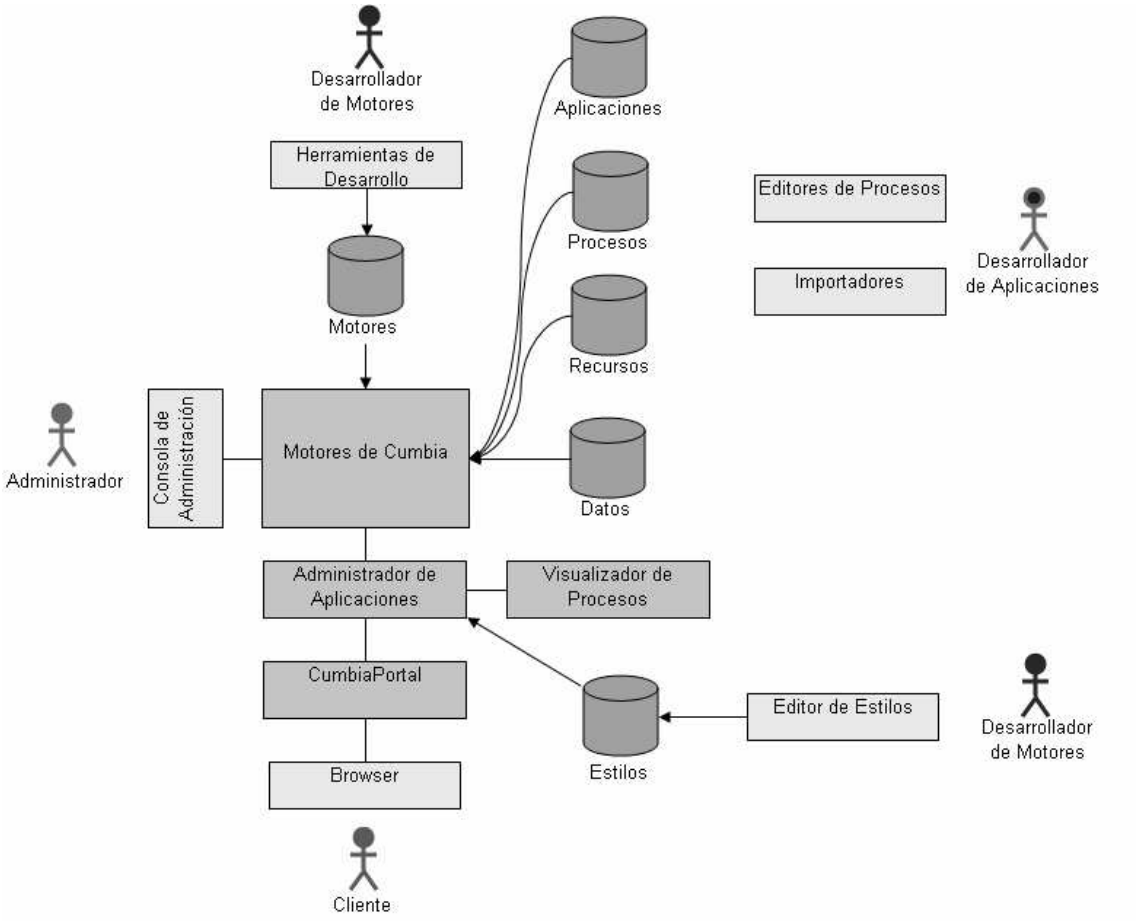

Figura No. 70: Arquitectura global de Cumbia

A continuación se describen cada uno de los componentes que se muestran en la arquitectura global de Cumbia.

5.1.1. Herramientas de desarrollo

Como se ha mencionado, uno de los objetivos de Cumbia es proveer una plataforma que sirva para construir motores de ejecución de procesos para diferentes lenguajes.

La construcción de un motor sobre cumbia consiste, básicamente, en dos tareas: (1) extender los elementos de CUMBIA-XPM, y (2) definir un lenguaje mediante la creación de un estilo.

La tarea de extender los elementos de CUMBIA-XPM se hace mediante los mecanismos de extensión de CUMBIA-XPM.

Para esto el desarrollador de motores usa las herramientas de desarrollo. Estas herramientas son editores para Java, tales como Eclipse, los cuales se utilizan para implementar clases Java para las acciones que pueden insertarse en las transiciones de los autómatas de los elementos de CUMBIA-XPM. También se necesitan editores para Java para extender los diferentes elementos de CUMBIA-XPM. Estas clases Java quedan almacenadas en un depósito de motores.

Se denomina depósito de motores porque un conjunto de clases forman un motor de ejecución de procesos específico.

5.1.2. Editores de procesos e Importadores

El desarrollador de aplicaciones tiene la función de construir las aplicaciones que se ejecutan sobre Cumbia.

Una aplicación de Cumbia consiste en la definición del proceso, la decoración de dicho proceso, un archivo descriptor de recursos y de reglas de asignación de recursos, y un archivo descriptor donde se especifica el estilo que se va a utilizar en la visualización del proceso.

Para diseñar un proceso, el desarrollador de aplicaciones utiliza un editor de procesos adecuado para el lenguaje que desee. El archivo de definición del proceso resultante es la entrada a un importador.

Un importador toma un proceso definido en un lenguaje específico y lo traduce a un proceso en Cumbia generando los archivos de definición de proceso y su respectiva decoración.

La siguiente figura muestra un ejemplo de un desarrollador de aplicaciones que utiliza el editor de procesos JaWE para crear un proceso XPDL, luego este proceso es recibido por el importador XPDL, el cual traduce dicho proceso generando un proceso Cumbia y su respectiva decoración.

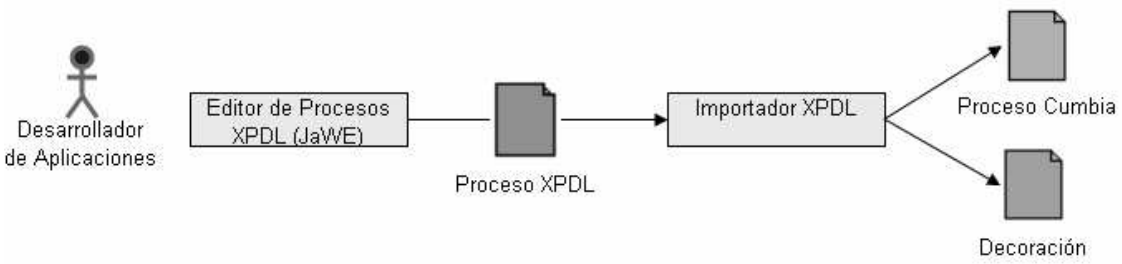

Figura No. 71: Ejemplo de importador XPDL

5.1.3. Editor de estilos

El desarrollador de motores también hace uso del editor de estilos para crear el estilo correspondiente al lenguaje del motor específico que se desea construir. El editor de estilos alimenta el depósito de estilos.

El editor de estilos es una aplicación Swing que permite crear, modificar y eliminar estilos de una forma amigable para el usuario.

5.1.4. Motores de Cumbia

El recuadro llamado Motores de Cumbia hace referencia al motor de recursos y al motor de procesos.

Motor de procesos

El motor de procesos es el encargado de ejecutar un proceso Cumbia. El motor de procesos recibe como entrada un archivo donde esta definido un proceso en XML, y otros archivos donde se definen los autómatas de los elementos de CUMBIA-XPM. Cada elemento de CUMBIA-XPM tiene su propio autómata. El motor, básicamente, es una máquina de ejecución de autómatas.

El motor de procesos también recibe como entradas clases Java de las acciones que se insertan en las transiciones de los autómatas de los elementos.

El motor de procesos provee una interfaz por medio de la cual se puede utilizar. Además, el motor de procesos genera eventos que son escuchados por sensores. Un sensor es un elemento que escucha un tipo específico de eventos. Los sensores son controlados por monitores. Un monitor toma los datos de los eventos que capturan los sensores, organiza estos datos y los comunica al interesado.

Como se ve en la siguiente figura, el administrador de aplicaciones utiliza los servicios del motor de procesos para iniciar

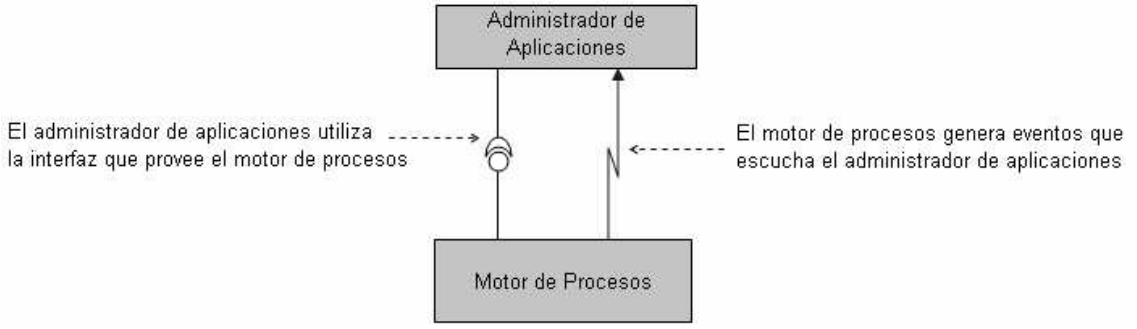

Figura No. 72: Motor de procesos y administrador de aplicaciones

Motor de recursos

El motor de recursos es el encargado de ejecutar las reglas de asignación de recursos que se definan para un proceso. La ejecución de estas reglas tiene como resultado la identificación de un recurso que va a ser asignado a un ítem de trabajo. Un ítem de trabajo puede ser una actividad, una multiactividad o un proceso.

El motor de recursos escucha eventos del motor de procesos, de esta forma el motor de recursos sabe cuando debe ejecutar cierto conjunto de reglas de asignación para retornarle al motor de procesos el recurso que se debe asignar al ítem de trabajo actual.

5.1.5. Administrador de aplicaciones

Este componente es un servidor de aplicaciones Cumbia. El administrador de aplicaciones es el encargado de lanzar o iniciar una aplicación y además, se encarga de administrar las aplicaciones que se están ejecutando sobre Cumbia en un momento dado.

Durante el despliegue de una aplicación Cumbia, el administrador de aplicaciones toma la definición del proceso y se la envía al motor de procesos, toma las reglas de asignación de recursos y se las envía al motor de recursos. El administrador de aplicaciones se puede ver como el intermediario entre los motores de Cumbia y los clientes. El administrador de aplicaciones se comunica con los motores para iniciar la ejecución de una aplicación. Los motores de Cumbia generan eventos, los cuales son recibidos por el administrador de aplicaciones. El administrador de aplicaciones decide que hacer con cada evento que recibe. Por cada aplicación Cumbia que se esta ejecutando, el administrador de aplicaciones mantiene un registro de los eventos que le interesa escuchar a cada aplicación. De esta forma si uno de los motores le envía un evento al administrador de aplicaciones, este determina a que aplicaciones debe enviarle este evento.

5.1.6. CumbiaPortal

CumbiaPortal es un componente que permite la interacción de los usuarios con Cumbia a través del web. Este componente es el encargado de: administrar las sesiones de los usuarios, pasar las peticiones de los usuarios al administrador de aplicaciones, y retornar las respuestas a los usuarios.

Una aplicación Cumbia puede tener varios módulos como: un módulo de visualización de procesos, un módulo de agenda con las actividades que debe ejecutar el usuario dentro del proceso, un módulo de datos donde el usuario puede ingresar o leer datos cuando las actividades lo requieren, y un módulo de recursos.

Estos módulos son visibles a través del web gracias a CumbiaPortal.

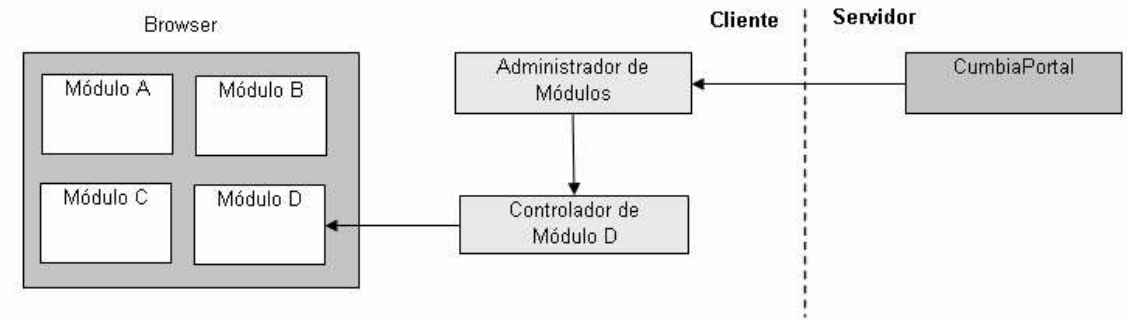

Figura No. 73: Módulos de CumbiaPortal

Del lado del servidor se encuentra CumbiaPortal, y del lado del cliente se encuentra el administrador de módulos, controladores de módulos y los módulos visualizados en el browser.

Cuando sucede un evento en el servidor, el evento llega a CumbiaPortal a través del administrador de aplicaciones. CumbiaPortal recibe el evento y lo envía a los clientes

adecuados. Cuando el cliente recibe una notificación que envió CumbiaPortal, esta notificación es recibida por el administrador de módulos. El administrador de módulos mantiene un registro de controladores de módulos por cada módulo que se visualiza en el browser. Cuando la notificación llega al administrador de módulos, este determina a cuáles módulos debe enviar el evento mediante su registro de controladores de módulos. Luego envía la notificación, y cada controlador de módulo sabe que hacer con la notificación, como actualizar los datos en el browser.

El envío de notificaciones desde el servidor hacia los clientes se hace por medio del protocolo http. Esto involucra la problemática de mantener sincronizados varios clientes con los eventos que ocurren en el servidor sin que los clientes tengan que hacer peticiones http para sincronizarse. Esta problemática se resuelve con la utilización de un framework basado en pushlets [Bro02].

Los *pushlets* son un mecanismo basado en *servlets*, donde los datos son enviados directamente desde el servidor hacia páginas HTML dinámicas (DHTML) sin usar *applets* de Java o cualquier otra tecnología como RMI. Este *framework* consiste en *servlets* y objetos Java en el servidor y una librería JavaScript en los clientes.

5.1.7. Visualizador de procesos

El visualizador de procesos llamado CumbiaViewer es el componente que permite visualizar gráficamente los procesos que se están ejecutando en Cumbia. CumbiaViewer esta basado en el modelo CUMBIA-XLM para generar los procesos gráficamente. Es por eso que CumbiaViewer debe acceder a los estilos, a la definición de los procesos y sus respectivas decoraciones.

CumbiaViewer es un EJB (Enterprise Java Bean) de sesión con un sólo método que recibe un evento. Los tipos de eventos son: instancia de proceso creada, cambio de estado de actividad, cambio de estado de puerto, y cambio de estado de *workspace*. De acuerdo al evento que recibe, CumbiaViewer actualiza el gráfico del proceso y retorna una respuesta. Esta respuesta contiene el identificador de la instancia del proceso y el URL de la imagen del proceso.

Tanto el editor de estilos como CumbiaViewer utilizan una librería llamada GumbiaGraph, la cual tiene los componentes necesarios para dibujar procesos a partir de la definición del mismo, la decoración y el estilo. CumbiaGraph esta construida sobre una librería de código abierto llamada Jgraph (www.jgraph.com).

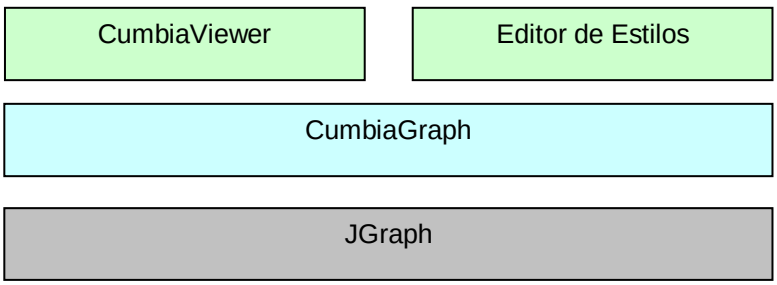

Figura No. 74: CumbiaViewer y Editor de Estilos

El administrador de aplicaciones se encarga de hacerle peticiones a CumbiaViewer cuando un cliente necesita visualizar el proceso en ejecución. Además, el administrador de aplicaciones le envía notificaciones a CumbiaViewer para que refresque la visualización de un proceso cuando dicho proceso cambia de estado.

5.1.8. Consola de administración

La consola de administración permite la administración de Cumbia. Entre las funciones de la consola de administración están: iniciar y finalizar la plataforma de Cumbia, modificar parámetros de configuración, visualizar el estado de los motores de Cumbia, y ver trazas de la ejecución de Cumbia.

5.2. Ejecución de un proceso

En esta sección se pretende mostrar la interacción entre los componentes de Cumbia, durante la ejecución de un proceso.

Un proceso inicia su ejecución cuando un cliente ingresa al portal de Cumbia y solicita la creación de una instancia de un proceso. El browser del cliente hace una petición http que es recibida por el portal de Cumbia CumbiaPortal.

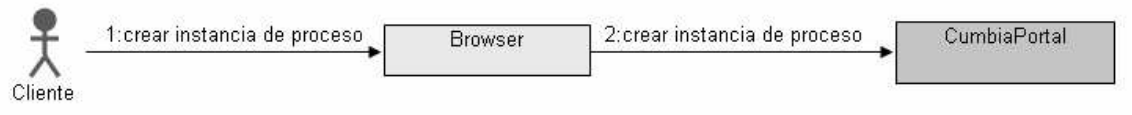

Figura No. 75: Ejecución de un proceso 1

La petición llega a CumbiaPortal, el cual, luego de verificar la sesión del usuario y sus privilegios, delega la creación de la instancia del proceso al administrador de aplicaciones. El administrador de aplicaciones se comunica con los motores de Cumbia para crear la instancia del proceso e iniciar su ejecución. El administrador de aplicaciones les proporciona los datos necesarios a los motores para iniciar la ejecución del proceso.

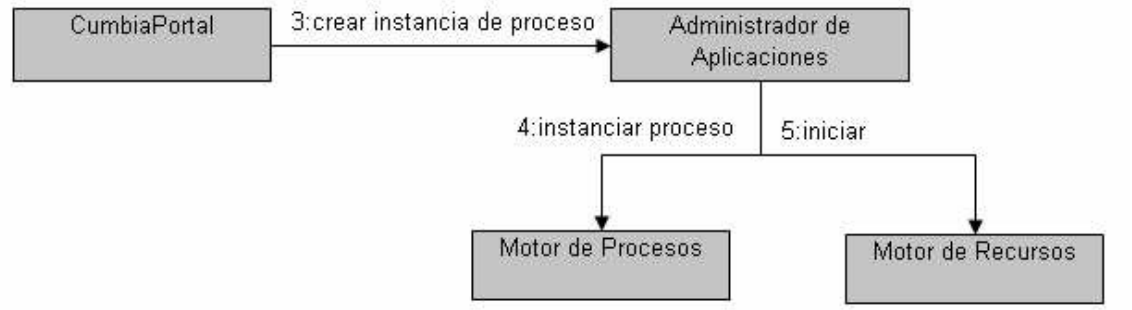

Figura No. 76: Ejecución de un proceso 2

Cuando el motor de procesos crea la instancia del proceso que se quiere ejecutar, este envía un evento indicando que se creó una instancia de un proceso. El evento

que se envía desde el motor de procesos tiene el nombre del proceso y el identificador de la instancia del proceso que se creó. Este evento lo recibe el administrador de aplicaciones. A partir del nombre del proceso, el administrador de aplicaciones ubica la definición de este proceso, la decoración, y el estilo con que se debe visualizar el proceso. Con estos datos, el administrador de aplicaciones interactúa con el visualizador de procesos CumbiaViewer para que el visualizador genere el proceso en forma gráfica.

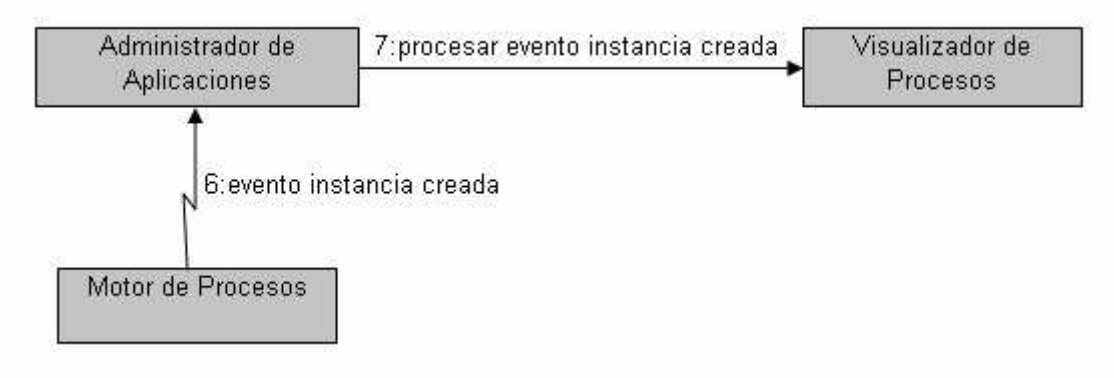

Figura No. 77: Ejecución de un proceso 3

CumbiaViewer recibe el llamado del administrador de aplicaciones. En este llamado tiene la definición del proceso, la decoración del proceso, y el estilo que se debe aplicar. CumbiaViewer toma estos datos y genera el gráfico del proceso. Luego, le responde al administrador de aplicaciones diciéndole que ya generó el gráfico del proceso y que ya puede ser desplegado en el cliente. Esta respuesta contiene el URL de la imagen del proceso.

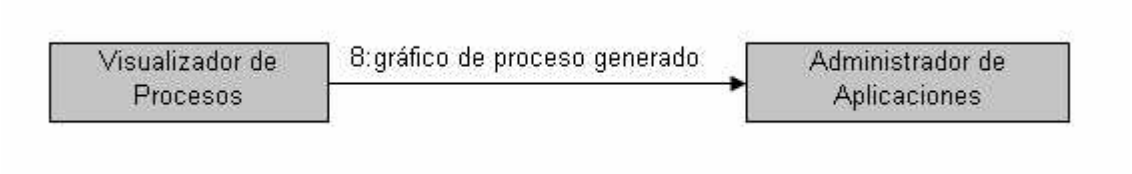

Figura No. 78: Ejecución de un proceso 4

El administrador de aplicaciones toma la respuesta de CumbiaViewer y envía un evento a CumbiaPortal con los datos necesarios para desplegar el proceso en el browser del cliente. Estos datos son: el identificador de la instancia del proceso, el identificador del usuario que debe recibir el evento, y el URL de la imagen del proceso.

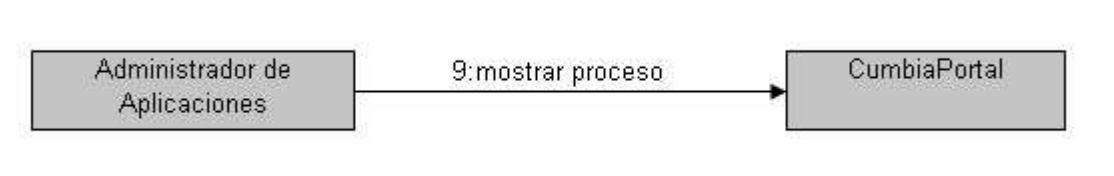

Figura No. 79: Ejecución de un proceso 5

CumbiaPortal recibe el evento que le envió el administrador de aplicaciones, determina a cuáles clientes debe enviarles la notificación y luego la envía. Del lado del cliente, hay un administrador de módulos. Este administrador recibe la notificación que envió CumbiaPortal, de acuerdo al tipo de notificación identifica a los controladores que debe

enviarle la notificación. El administrador de módulos identifica que debe enviarle la notificación al controlador del módulo de visualización y le pasa la notificación. El controlador del módulo de visualización de procesos toma el evento y actualiza la vista del proceso.

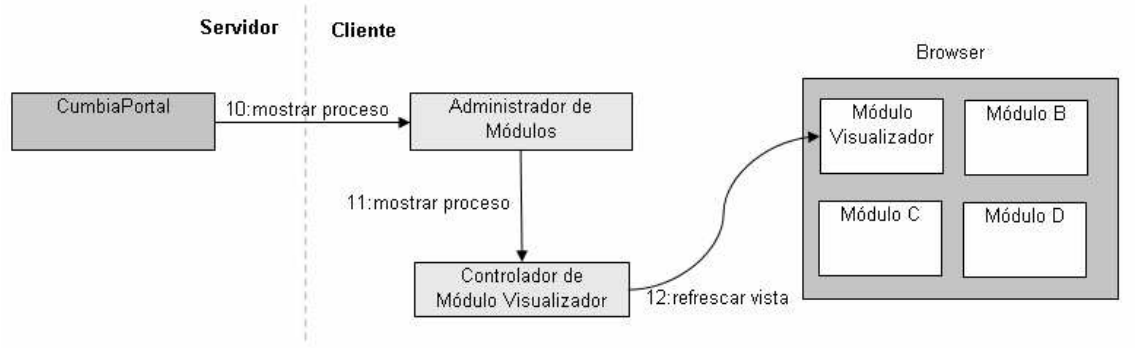

Figura No. 80: Ejecución de un proceso 6

Hasta este punto sólo se ha creado una instancia de un proceso para ser ejecutada. A partir de aquí, lo que continúa pasando es que se empiezan a ejecutar las actividades del proceso. Cuando una actividad comienza a ejecutarse el motor de procesos genera un evento notificando el cambio de estado de la actividad. Este evento genera una interacción similar a la que se mostró anteriormente cuando el motor genera el evento de instancia de proceso creada. La siguiente figura muestra la interacción cuando se genera el evento de cambio de estado de una actividad.

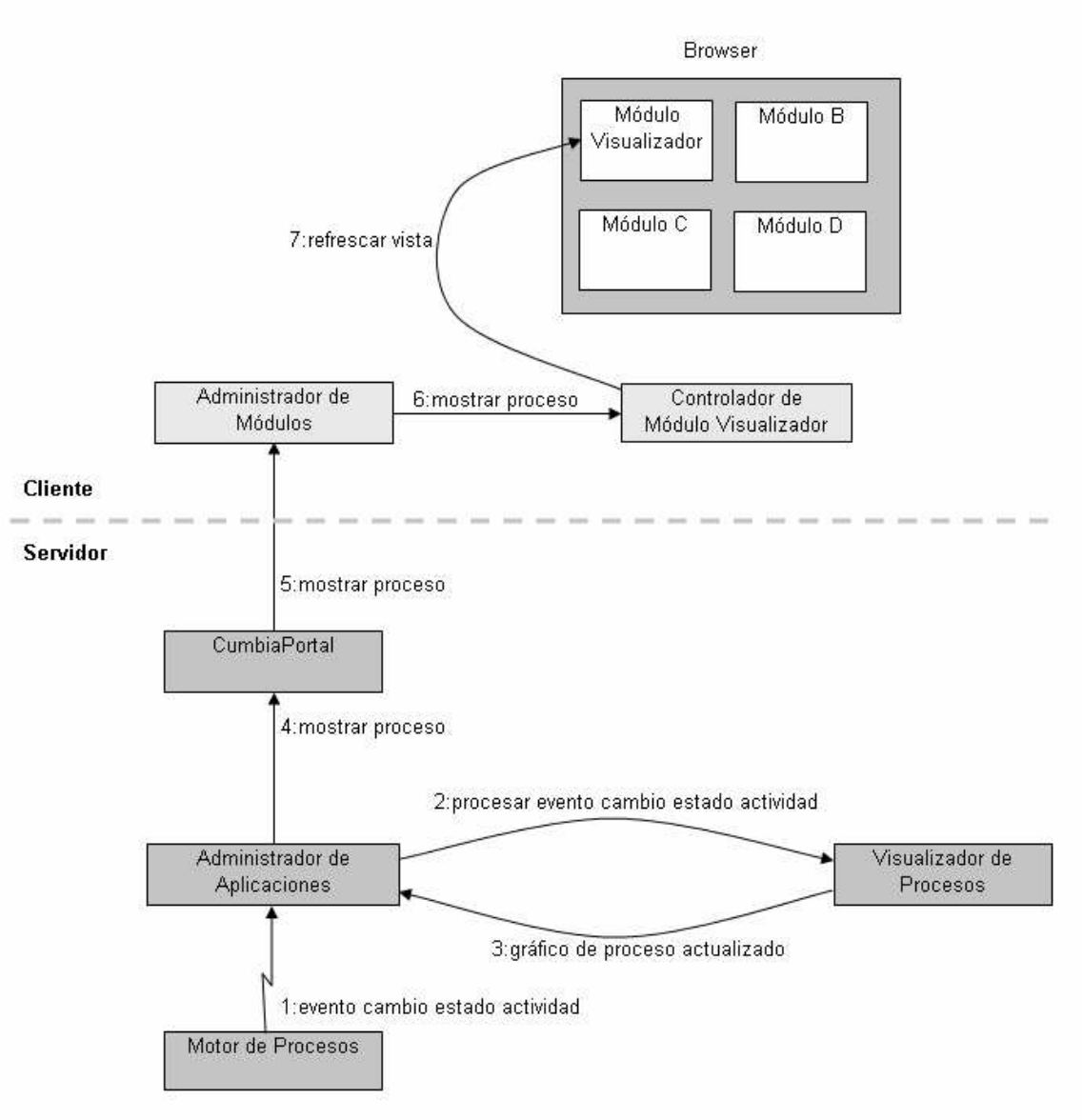

Figura No. 81: Ejecución de un proceso 7

6. Conclusiones

En este capítulo se presentan los resultados obtenidos y los trabajos futuros.

6.1. Resultados Obtenidos

La intención de Cumbia es proporcionar un motor de motores de ejecución de procesos. Motor de motores porque a partir de Cumbia se pueden construir motores de ejecución de procesos.

Sin embargo, Cumbia no tiene su propio lenguaje de definición de procesos. La razón de este hecho es que sobre Cumbia se pretenden crear otros motores de ejecución de procesos y cada uno de estos motores tendrá su propio lenguaje. Debido a esto es necesario tener un mecanismo para poder crear diferentes lenguajes para los diferentes motores.

Inicialmente, se pensó en una forma de asignar grafismos a los elementos del modelo CUMBIA-XPM. Se buscaba que la creación y asignación de grafismos mediante un documento XML. De esta forma podría implementarse un intérprete que fuera capaz de dibujar procesos con los grafismos que se describen en el documento XML.

Esto dio como resultado un modelo para crear lenguajes de definición de procesos. Este modelo se desarrolló pensando en que tuviera el mínimo de elementos posibles y los suficientes mecanismos de extensión para soportar la creación de diferentes lenguajes. Este modelo se llama CUMBIA-XLM (Cumbia Extensible Language Model).

Cuando se terminó de definir y estructurar todos los elementos del modelo CUMBIA-XLM, el siguiente paso fue crear la especificación del modelo. Esta especificación consiste de una sección donde se presentan los elementos de un estilo, donde para cada elemento se muestra como esta compuesto y se exponen las invariantes sobre estos elementos. Otra sección de la especificación muestra los elementos de la decoración. Dos secciones más, muestran cómo se deben definir el estilo y la decoración en XML. Luego se presentan algunos ejemplos para el estilo y la decoración. Por último, la especificación tiene algunos diagramas de clases que muestran las asociaciones entre los elementos del modelo.

El siguiente paso del trabajo de esta tesis consistió en crear los estilos para algunos de los principales lenguajes y modelos que existen actualmente. Esto con el fin de analizar la aplicabilidad del modelo CUMBIA-XLM. De esta actividad se detectaron algunas limitaciones del modelo pero que no son graves; y se concluyo que el modelo provee los mecanismos suficientes para crear lenguajes reales.

Una vez se tuvo la especificación del modelo en una versión estable, se procedió a implementar los componentes necesarios para la visualización de procesos con estilos. De esta implementación se obtuvo un framework base llamado CumbiaGraph, sobre el cual se desarrolló el visualizador de procesos CumbiaViewer y el editor de estilos.

Los resultados de este trabajo de tesis fueron los siguientes:

- Se creó un modelo de definición de lenguajes minimal y extensible.

- Se creó una especificación del modelo CUMBIA-XLM, en la cual se presentan todos los detalles del modelo.
- Se construyeron estilos para los principales lenguajes y modelos que se encuentran actualmente en el mercado, mostrando que CUMBIA-XLM tiene los elementos necesarios para soportarlos.
- Se desarrollaron los componentes relacionados con la visualización de procesos, con la capacidad de ser integrados a la arquitectura de Cumbia.
- Se desarrolló un editor de estilos que facilita la creación de estos.

Los resultados obtenidos son un aporte al proyecto Cumbia. En pocas palabras, el aporte más importante de esta tesis es el modelo de definición de lenguajes CUMBIA-XLM para Cumbia.

6.2. Trabajos Futuros

En cuanto al modelo CUMBIA-XLM, se considera que es interesante y de gran aporte el trabajar alrededor de los siguientes puntos:

6.2.1. Demostración formal de la capacidad de CUMBIA-XLM

En este documento se evaluó la capacidad de definir lenguajes por medio de CUMBIA-XLM creando estilos para los principales lenguajes y modelos existentes. Sin embargo, es necesario demostrar la capacidad de crear diferentes lenguajes con CUMBIA-XLM a través de un método formal. Esta demostración debe garantizar que si CUMBIA-XPM soporta un modelo de procesos MP, entonces CUMBIA-XLM soporta la creación del lenguaje para el modelo MP.

6.2.2. Importadores

Un importador toma un proceso definido en un lenguaje de definición de procesos L y lo traduce o importa a Cumbia generando un documento XML que describe el proceso en Cumbia y otro documento XML, el cual es la decoración del proceso. De esta forma, un importador depende del lenguaje de definición de procesos L desde el cual se importan los procesos.

Los importadores realizan un proceso de traducción que es cableado en la implementación del mismo. Sin embargo, algunos elementos de un lenguaje pueden ser traducidos a Cumbia de más de una forma y otros elementos simplemente pueden no tener una traducción directa a Cumbia.

Definir las reglas de traducción es un proceso complejo y resulta interesante trabajar en esta problemática considerando técnicas de traducción asistidas por el usuario.

6.2.3. Editor genérico de procesos

Un gran aporte al proyecto Cumbia que involucra el modelo CUMBIA-XLM es desarrollar un editor genérico de procesos. Este editor, a partir de un estilo, permite editar procesos de Cumbia utilizando el lenguaje definido en dicho estilo. Es decir, que el editor tiene la capacidad de crear procesos en diferentes lenguajes siempre y cuando cada uno de estos lenguajes tenga su respectivo estilo.

Esta es una tarea muy compleja pero que significaría un gran aporte a Cumbia, ya que con un único editor se pueden crear procesos para todos los motores que se construyan sobre Cumbia.

6.2.4. Creación de estilos para otros lenguajes

En esta tesis se crearon estilos para los principales lenguajes y modelos que existen. Sin embargo, es un buen aporte crear los estilos para otros lenguajes, de tal forma que se tenga un catalogo de estilos lo más completo posible y así Cumbia tenga soporte para estos lenguajes.

6.2.5. Mejoras al editor de estilos

El editor de estilos permite crear y modificar estilos. Este editor posee las funciones necesarias para la edición de los estilos. Sin embargo, el editor de estilos se puede enriquecer con funciones adicionales como Copiar – Pegar, Deshacer – Rehacer. Además de convertir el editor en un plugin de eclipse.

El aporte de este trabajo radica en mejorar la facilidad de uso del editor de estilos, para que los usuarios puedan crear estilos con mayor facilidad.

7. Referencias Bibliográficas

[Aal03] W. van der Aalst. “Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language”. Department of Technology Management Eindhoven University of Technology, The Netherlands. 2003.

[ABHK00] W. van der Aalst, A. Barros, A. Hofstede, B. Kiepuszewski. “Advanced Workflow Patterns”. 2000.

[ADHW03] W. van der Aalst, Marlon Dumas, Arthur H.M. ter Hofstede, Petia Wohed. “Pattern Based Análisis of BPML (and WSCI)”. 2003

[AHKB00] W. van der Aalst, A. Hofstede, B. Kiepuszewski, y A. Barros. “*Workflow Patterns*”. 2000.

[AC05] Paul Andrew, James Conrad. “Presenting Windows Workflow Foundation”. Beta Edition, Sams. 2005

[ACDGKLLRSTTW03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana. “Business Process Execution Language for Web Services”. BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems. 2003

[Bay05] Tom Bayens. “Java Business Process Management”, Version 3.0. Disponible en: http://www.jbpm.org/3/. 2005.

[BMPS05] Bojanic S, Milakovic Z, Puskas V, Stefanovic N. JaWE documentation. Disponible en http://jawe.objectweb.org/doc/1.4/index.html

[BT98] Gregory Alan Bolcer, Richard N. Taylor. “Advanced Workflow Management Technologies”. 1998

[Bro02] Just van den Broecke. “Pushlets Whitepaper”. Just Objects. 2002

[Esp05] Dino Esposito. “Getting Started with Microsoft Windows Workflow Foundation: A Developer Walkthrough”. Microsoft Corporation. 2005

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. “Design Patterns: Elements of reusable Object-Oriented Software”. Addison-Wesley. 1995.

[IMS03] IMS Global Learning Consortium, Inc. “IMS Learning Design Information Model”. Version 1.0 Final Specification. 2003

[Intalio] Intalio|n^3. www.intalio.com. 2005

[LS97] Yu Lei, Munindar P. Singh. “A Comparison of Workflow Metamodels”. Department of Computer Science North Carolina State University Raleigh. 1997.

[Ley01] Frank Leymann. “Web Services Flow Language (WSFL 1.0)”. IBM Software Group. 2001.

[MCVCC05] Olga Mariño, Rubby Casallas, Jorge Villalobos, Dario Correal, Julián Contamines. “Bridging the Gap between E-learning Modeling and Delivery through the Transformation of Learnflows into Workflows”. 2005.

[Ped05] Gabriel R. Pedraza. “Cumbia Extensible Process Modeling:

Un Modelo Extensible para la Construcción de Motores de Workflow y su Validación Mediante Representación de BPEL4WS, XPDL y JBPM". Universidad de los Andes, Colombia. 2005.

[Ped05b] Gabriel Pedraza. "Representación de las principales construcciones de BPEL4WS, XPDL y JBPM en Cumbia-XPM". Universidad de los Andes, Colombia. 2005

[PS05a] Gabriel Pedraza, Jaime Solano. "Comparación de Modelos de Workflow: BPEL, XPDL, BONITA, jBPM y APEL". Universidad de los Andes, Colombia. 2005.

[PS05b] Gabriel Pedraza, Jaime Solano. "Análisis de APEL, Basado en Patrones de Flujo". Universidad de los Andes, Colombia. 2005.

[RHEA05] Nick Russell, Arthur H.M. ter Hofstede, David Edmond, W. van der Aalst. "Workflow Resource Patterns". 2005.

[Sol05a] Jaime Solano. "Cumbia Extensible Process Modeling: Un Modelo Extensible para la Construcción de Motores de Workflow y su Validación Mediante Patrones de Control de Flujo". Universidad de los Andes, Colombia. 2005.

[Sol05b] Jaime Solano. "Análisis de CUMBIA-XPM, Basado en Patrones de Control de Flujo". Universidad de los Andes, Colombia. 2005.

[SPEM05] "Software Process Engineering Metamodel Specification". Object Management Group. 2005.

[Tha01] Satish Thatte. "XLANG Web Services for Business Process Design". Microsoft Corporation. 2001. Disponible en: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[Vil02] Jorge Villalobos, APEL Light Specification. LSR-IMAG, Francia, 2002.

[VTCNCSPM05] Jorge Villalobos, Silvia Takahashi, Dario Correal, Carlos Noguera, Rubby Casallas, Jaime Solano, Gabriel Pedraza, David Murillo. "Cumbia-XPM: Cumbia Extensible Process Modelling". Universidad de los Andes, Colombia. Octubre, 2005.

[VTCNPSMC05] Jorge Villalobos, Silvia Takahashi, Darío Correal, Carlos Noguera, Gabriel Pedraza, Jaime Solano, David Murillo, Alex Chacón. "CUMBIA-XLM: Extensible Language Model". Universidad de los Andes, Colombia. Octubre, 2005.

[Whi04] Stephen A. White. "Introduction to BPMN". IBM Corporation. 2004

[WMC01] Workflow Management Coalition. "Workflow Process Definition Interface – XML Process Definition Language" (XPDL) (WFMC-TC-1025). 2001.

[WMC95] Workflow Management Coalition. "The Workflow Reference Model" (WFMC-TC-1003). 1995.

[WADH03] Petia Wohed, Will M.P. van der Aalst, Marlon Dumas, Arthur H.M. ter Hofstede. "Analysis of Web Services Composition Languages: The Case of BPEL4WS". 2003.