El/la abajo firmante, Laura Henche Grande, matriculado/a en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Introducción a la notación BPMN y su relación con las estrategias del lenguaje Maude", realizado durante el curso académico 2008-2009 bajo la dirección de Narciso Martí Oliet en el Departamento de Sistemas Informáticos y Computación, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

MASTER EN INVESTIGACIÓN EN INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

PROYECTO FIN DE MASTER EN PROGRAMACIÓN Y TECNOLOGÍA SOFTWARE

CURSO 2008/2009

INTRODUCCIÓN A LA NOTACIÓN BPMN Y SU RELACIÓN CON LAS ESTRATEGIAS DEL LENGUAJE MAUDE

Autor: Laura Henche Grande

Director: Narciso Martí Oliet

Resumen

El proyecto es un estudio sobre el modelado de procesos de negocio (BPM) donde se describen las partes en las que debe componerse una arquitectura o cómo hacer uso de patrones durante el diseño de los procesos. También analiza las técnicas y estándares más destacados que hay en la actualidad. Por otra parte, del lenguaje Maude se hace una introducción a los tipos de módulos necesarios para poder definir especificaciones de datos. Al final ambos conceptos, BPM y Maude, se ven relacionados a través de la notación gráfica BPMN y de las estrategias de Maude, que cómo se verá son dos maneras de controlar el comportamiento de los procesos. Nuestra propuesta es realizar una aproximación a través de la correspondencia entre los símbolos de la notación y las estrategias que dirigen las reglas del sistema. Se incluye un ejemplo sobre catalogación y consulta de artículos donde se puede ver esta relación, llamado Portal de Revistas Complutenses.

Palabras clave

Arquitectura BPMN diagrama estrategia Maude modelo negocio notación patrón proceso

ABSTRACT

This is a project about the business process modeling (BPM), where the different parts that any architecture should contain are described. It is also explained how to use the patterns during the design of the process and the most common techniques and standards which we have nowadays. Besides there is an introduction to the types of modules that we need to define our data specifications. In the end both concepts, BPM and Maude, are related by the graphic notation BPMN and the Maude strategies, which conform, as we will see, two different options to control the behavior of the process. Our intention is to relate them through the correspondence between the symbols of the notation and the strategies which lead the rules of the system. It is also included an example called “Portal de Revistas Complutenses” about catalog and search of articles where this relationship can be observed.

KEYWORDS: architecture BPMN business diagram Maude model notation pattern process strategy

ÍNDICE

Introducción a los modelos de procesos de negocio

¿Por qué existen procesos de negocio? Toda organización tiene un propósito y para conseguir este objetivo eficientemente, el trabajo se divide en un número de procesos, o "*formas de hacer las cosas de una manera repetitiva*", que trabajan juntos para contribuir en la organización. Cada uno de estos procesos tiene su propósito propio y responsabilidades que contribuyen a los objetivos generales.

El motivo para querer hacer estos procesos consistentes es porque hacer lo mismo cada vez lo vuelve más eficiente y siempre hay menos posibilidades de cometer un error, además la experiencia permite refinar el proceso tomando en cuenta situaciones que pueden ligeramente salirse de lo normal.

Por *proceso de negocio* se entiende el conjunto ordenado de actividades dentro de una empresa, cada una de las cuales representa el trabajo de una persona, sistema interno o proceso de una compañía asociada. Las *actividades* son la unidad más pequeña de análisis, y su orden define el flujo de trabajo que determina la eficiencia de la estructura.

Entonces, ¿Qué es un modelo de procesos de negocio? El término de *modelado de proceso de negocio* (BPM) define el soporte usado para incorporar las actividades de los sistemas de negocio en modelos que describen los procesos ejecutados por la organización. Incluye un conjunto de técnicas y estándares para el diseño, ejecución, administración y monitorización de procesos de negocio. En los primeros capítulos se explica cómo realizar estos modelos de procesos de negocio, y cómo hacer uso de patrones que ayuden durante su definición. También se describen las técnicas y estándares definidos actualmente.

Adoptar BPM fuerza al negocio a pensar sobre él y a formalizarlo comprendiendo los procesos actuales, lo que hace ver mejoras potenciales que eliminan pasos o automatizan tareas manuales. El propósito de los modelos de negocio por tanto es cuestionar la forma en que se hacen las cosas, y cómo se podrían querer hacer en el futuro.

Para crear un modelo de proceso de negocio, primero necesitamos comenzar por definir qué proceso nos permite lograr el objetivo, y cómo encaja con el resto de procesos de la organización. La mejor visualización de un proceso es a través de diagramas que muestran el flujo de trabajo, indicando las actividades que deben ser visitadas a lo largo de la ejecución y qué necesita hacerse cuando una excepción ocurre. Para ello explicaremos con detenimiento el estándar BPMN en el sexto capítulo.

Una vez entendido en que consisten los modelos de procesos de negocio, el objetivo principal del proyecto es ver cómo encaja el lenguaje Maude en este ámbito, el cual está descrito de manera introductoria en el capítulo siete. Para ello se ha estudiado su relación con la notación BPMN, comprobando si es posible a partir de sus diagramas generar código con el uso del lenguaje de estrategias de Maude. Su demostración se ha realizado sobre un ejemplo que consiste en la catalogación y acceso a artículos digitalizados, al que hemos puesto el nombre de Portal de Revistas Complutenses, y que es analizado en el octavo capítulo. En el apéndice se incluyen los módulos implementados.

Cómo diseñar una arquitectura BPM

Para diseñar una buena solución BPM, el primer paso es examinar el entorno del proyecto para comprender el problema y establecer qué procesos van a ser modelados. Luego el analista de negocio debe diseñar cada proceso, describiendo el camino que debe llevar: cómo empieza, qué ejecuta y cuál debe ser su resultado. Después el proceso es ejecutado por un motor, al que se le suministra la lógica que fue definida. El proceso de negocio a menudo requiere participación humana, es por ello que el motor debe tener capacidad de interacción con las personas. Además se debe dar soporte a la gestión y coordinación de las comunicaciones entre los sistemas.

La gestión de los procesos de negocio tiene normalmente cuatro fases: diseño, configuración, implementación y optimización. En la fase de diseño se requieren modelos cuyo propósito sea describir a los procesos, para que su conocimiento sea capturado y analizado. En la fase de configuración, el modelo es colocado en el entorno actual. En la fase de implementación el proceso se vuelve ejecutable, y lo que se requiere son modelos cuyo propósito sea soportar decisiones para monitorizar y controlar a los procesos. En la fase de optimización, se usan métodos que descubran las ventajas y desventajas de los procesos para mejorar.

Al final del desarrollo la arquitectura BPM debe componerse de las siguientes partes:

- **Diseño**: el diseño define el comportamiento o algoritmo básico del proceso. Consiste en un diagrama de flujo que resume los pasos ejecutados en cada momento para la resolución de un problema de negocio. El analista de negocio debe verse involucrado en esta fase porque él es el que comprende los aspectos de negocio del proceso. Por otro lado, los analistas técnicos son los que dibujan la previsión de la implementación de la solución software. Es por eso que los diseños de negocio y técnicos requieren de una notación gráfica común, que al mismo tiempo oriente en el diseño y sea flexible en el procesamiento computacional.

  - *Notación*: el diseño se transmite mejor con diagramas que con palabras. Y si además el diagrama se dibuja con una especificación estándar se le añade más

alcance, porque al ser conocido por una amplia audiencia, su semántica es comprendida con mayor claridad. BPM tiene dos notaciones principales para el modelado gráfico: *BPMN* y los *diagramas de actividad* de UML.

- **Ejecución**: para la ejecución de los procesos se requiere de una máquina capaz de ejecutar lo que se ha dibujado durante el diseño. Es por ello que se necesita de un mapeo entre la notación del diseño y el lenguaje de ejecución que la máquina conoce, que permita la generación automática de código ejecutable a partir del diseño. No se quiere la antigua codificación manual de los procedimientos porque ralentiza el desarrollo y asegura diferencias entre el diseño y la ejecución.

  - La *máquina BPM* carga programas con los diseños, llamados definición de procesos, y ejecuta instancias de ellos, llamadas procesos. El *programa BPM* es una serie de pasos, y el trabajo de la máquina es ejecutar los procesos a través de esos pasos, al igual que un procesador ejecuta los programas a través de instrucciones en líneas de código. El programa espera a que el siguiente evento ocurra y reacciona ante él ejecutando las actividades adecuadas. Pero es la máquina la responsable de la gestión de los eventos en los procesos, detectándolos y generándolos en cada momento. Respecto a los estándares, *BPEL* es el mejor y más ampliamente adoptado en lenguajes de ejecución.

- **Monitorización y administración**: la *monitorización* es la capacidad para ver el progreso de los procesos ejecutados y es crucial para las aplicaciones por dos razones: la detección de excepciones, asegurando que los procesos están progresando como esperamos, y las preguntas en tiempo real. La *administración* es la gestión de los procesos y la habilidad para cambiar sus efectos en la máquina. Debe permitir iniciar, parar o instalar nuevas definiciones de procesos, y suspender, continuar y terminar con la ejecución de las mismas. La monitorización y administración de procesos requiere de una interfaz donde se pueda configurar el sistema BPM.

- **Interacción humana**: una tarea manual es un paso en un proceso que es ejecutado por una persona concreta o una persona con un rol específico dentro de

la compañía. Lo normal es que varias personas participen en el mismo proceso y la misma persona intervenga en varios procesos simultáneamente. Para soportar el procesamiento manual, la arquitectura requiere de: un modelo que asocie usuarios con roles, una interfaz que muestre al usuario la lista de las actividades manuales que tiene asignadas y le permita completar las tareas pendientes, y un proceso de coordinación con la máquina que permita el intercambio de información.

- **Interacción con el sistema**: además de las dependencias humanas, hay otros pasos en el proceso que deben ser capaces de llamar o ser llamados por componentes software que no han sido generados por la máquina. Estos componentes pueden ser conexiones con participantes externos o sistemas internos a la compañía. Las interacciones externas son servicios que siguen protocolos de colaboración. Las interacciones con software interno pueden ser fragmentos de líneas de código o interfaces con otras aplicaciones que se ejecutan en la red de la corporación. La arquitectura debería soportar interfaces de entrada y salida para las tecnologías más comunes, como los servicios web o XML y suministrar un plugin para el resto.

  - *Coreografía contra orquestación*: la mayoría de los lenguajes de ejecución representan aisladamente a cada uno de los participantes, centrándose en sus procesos internos e interacciones, lo que le hace ser fundamentalmente *subjetivo*. La visión *objetiva* es distinta: la lógica interna de cada proceso es indiferente, lo que importa es el protocolo que se debe cumplir para la comunicación con el resto de participantes externos. Coreografía y orquestación representan estos dos puntos de vista: la orquestación describe el control central del comportamiento como un director en una orquesta, mientras que la coreografía trata sobre el control distribuido del comportamiento, como en una danza coreográfica. O dicho de otro modo, la orquestación es la coordinación a nivel *local* de un único proceso participante, mientras que la coreografía es la perspectiva *global* de colaboración entre los múltiples participantes. En coreografía *WS-CDL* es el mejor de los estándares.

Diseño de Patrones

Un diseño de patrón es una solución a un problema común, con el objetivo de ayudar al diseñador a la hora de combinar las actividades. El *patrón* lo que hace es proponer un conjunto de actividades organizadas de la mejor forma para resolver un tipo de problema. Es por eso que el uso de patrones en BPM permite construir mejores procesos.

A continuación se describe un listado de patrones específicos para el flujo de control de los procesos.

Patrones básicos

- Secuencia: consiste en ejecutar actividades ordenadas secuencialmente. Por ejemplo, ejecutar la actividad A seguida por B seguida por C, y así sucesivamente.

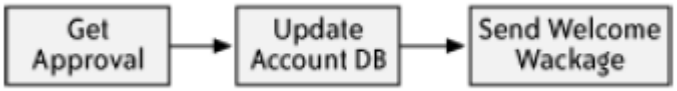


Ilustración 1: Patrón de secuencia

- División paralela: consiste en ramificar una única actividad en múltiples rutas paralelas. Tiene una sola entrada y dos o más salidas. Es también conocido como *AND-split*.

- Unión paralela: consiste en tener varias rutas paralelas que convergen en una única actividad, la cual espera a la finalización de todas las rutas para comenzar. Es la unión de las rutas paralelas producidas por la división paralela. Tiene dos o más entradas y una única salida. También se conoce como *AND-join*.

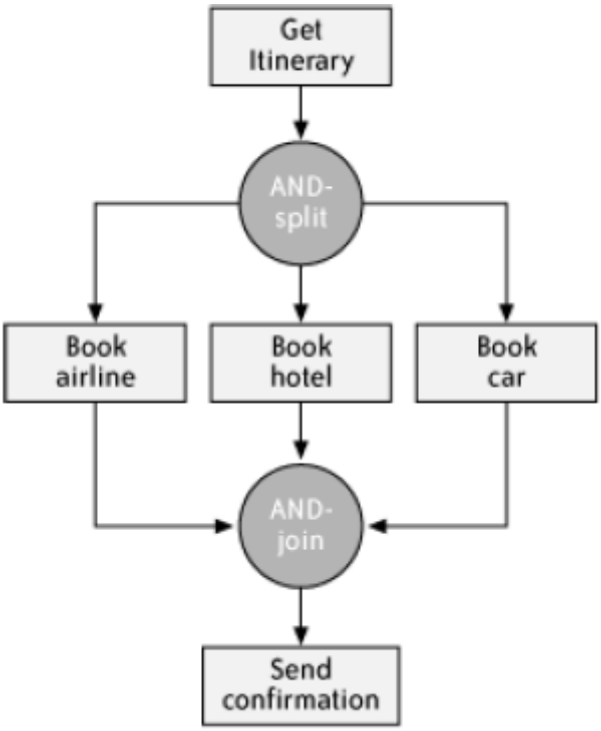


Ilustración 2: Patrones de división y unión paralela

En el diagrama cuando se completa la actividad de elección de itinerario, las tres reservas comienzan a ejecutarse en paralelo. El uso del paralelismo en este ejemplo es debido a que las rutas son independientes, es decir, si un retraso ocurre en alguna de las reservas, las otras no se ven afectadas. La actividad de envío de confirmación no se realiza hasta que las tres reservas precedentes han sido completadas.

- Elección exclusiva: consiste en elegir, desde una actividad, una de entre varias rutas, basándose en la evaluación de una condición. El efecto es el de una instrucción if en el proceso. Tiene una sola entrada y dos o más salidas. También se le llama *XOR-split*.

- Unión exclusiva: consiste en que varias rutas iniciadas por una elección exclusiva convergen en una única actividad, la cual comienza cuando la ruta elegida ha sido completada. Puede ser considerado como el final de una instrucción if. Tiene dos o más entradas y una única salida. Se le llama también *XOR-join*.

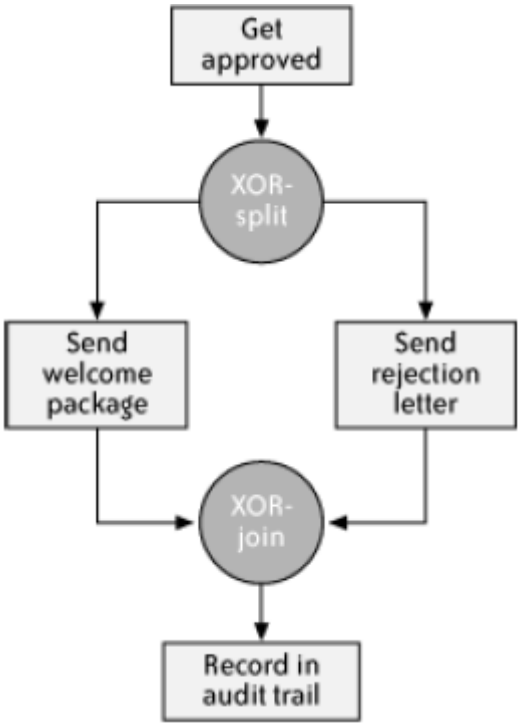


Ilustración 3: Patrones de elección y unión exclusiva

Este es un ejemplo donde si una petición es aprobada se da la bienvenida al cliente y si es rechazada se le envía una notificación, pero solo una de las dos opciones puede ocurrir.

Patrones avanzados de división y unión

- Elección múltiple: consiste en la bifurcación de un proceso en múltiples rutas, que están basadas en una condición. Se sigue en paralelo por aquellas rutas que satisfacen su condición particular. También es conocido por *OR-split*.

- Unión sincronizada: consiste en unir las rutas producidas por la elección múltiple, para ello espera a que todas las rutas activas en paralelo se hayan completado. Se le conoce por *OR-join*.

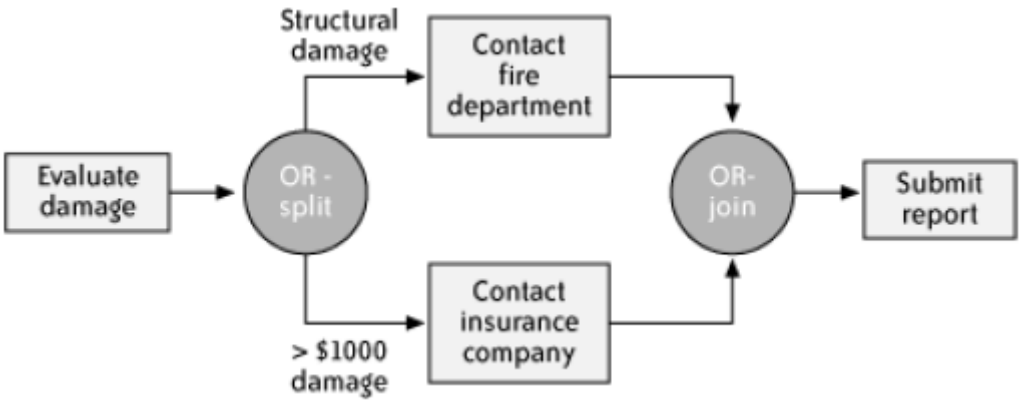


Ilustración 4: Patrones de elección múltiple y unión sincronizada

En este ejemplo la compañía de seguros o el departamento de bomberos o ambos deben ser contactados. La elección se basa en las condiciones específicas del fuego, es decir, se avisa a los bomberos si está dañada la estructura del edificio y se contacta con el seguro si se estiman pérdidas superiores a los 1000 dólares.

- Unión múltiple: consiste en que cada ruta de una bifurcación continua independientemente de las otras, permitiendo múltiples hilos de ejecución. Por ejemplo, cuando dos actividades concurrentes A y B, son multi-mezcla con C, las posibles combinaciones de ejecución son: ABCC, ACBC, BACC, BCAC. También es conocido por *unión descontrolada*. La mayoría de los lenguajes no permiten que la misma actividad sea ejecutada por hilos separados, y proponen una implementación alternativa donde cada ruta ejecuta una copia.

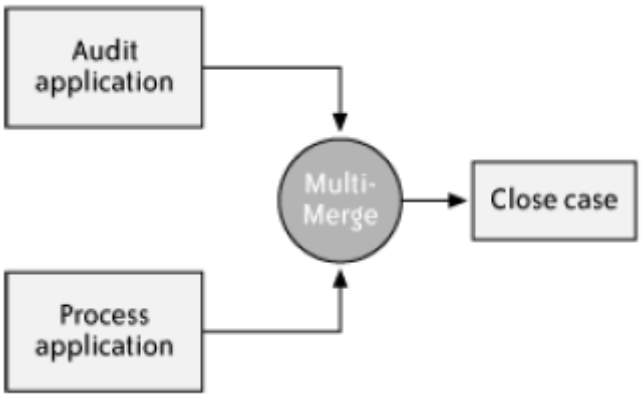


Ilustración 5: Patrón de unión múltiple

- Unión N de M: cuando múltiples rutas convergen en un punto, sólo las N primeras pueden continuar con el proceso, basándose en una condición evaluada; el resto de rutas son bloqueadas. También se le denomina *unión compleja*.

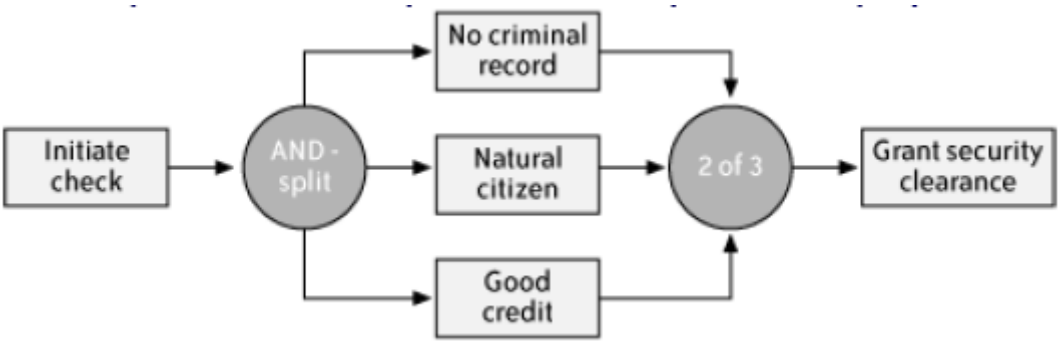


Ilustración 6: Patrón 2 de 3

En el ejemplo para obtener la autorización deben aceptarse dos de las tres condiciones. Cuando dos de las actividades se completan, la tercera es descartada.

- Discriminación: es un caso especial de patrón unión N de M donde sólo a la primera ruta se le permite continuar con el proceso.

Patrones estructurales

- Ciclos arbitrarios: consiste en repetir una actividad o un conjunto de actividades en el proceso. También es conocido como *GOTO* o *bucle*.

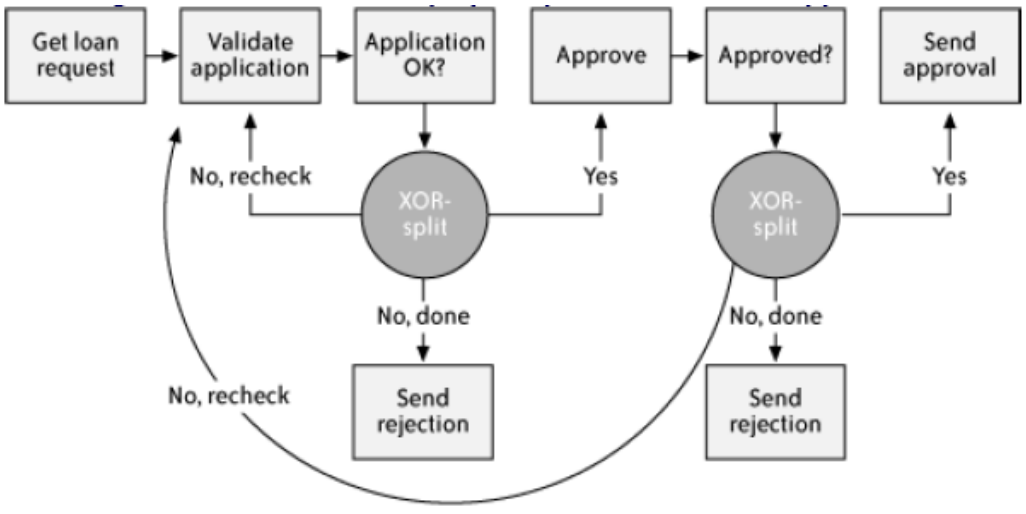


Ilustración 7: Patrón de ciclos arbitrarios

Este proceso está diseñado para retroceder cuando la validación falla o algo ha evitado la aprobación.

- Terminación implícita: en la mayoría de los lenguajes, un proceso tiene un único punto de salida en el cual todas las posibles rutas convergen. Para ayudar a reducir la complejidad en un proceso, este patrón termina el proceso cuando las actividades de cada una de las rutas han sido completadas, suavizando la restricción de que todas las rutas acaben en un mismo punto de salida.

Patrones basados en estados

Los siguientes patrones se aplican a procesos que son conducidos por eventos, por lo que esperan a que ocurra uno para lanzar la siguiente actividad.

- Elección aplazada: consiste en elegir entre una de múltiples rutas, que es aplazada hasta que ocurre el respectivo evento asignado. Es también conocido por *Pick*.

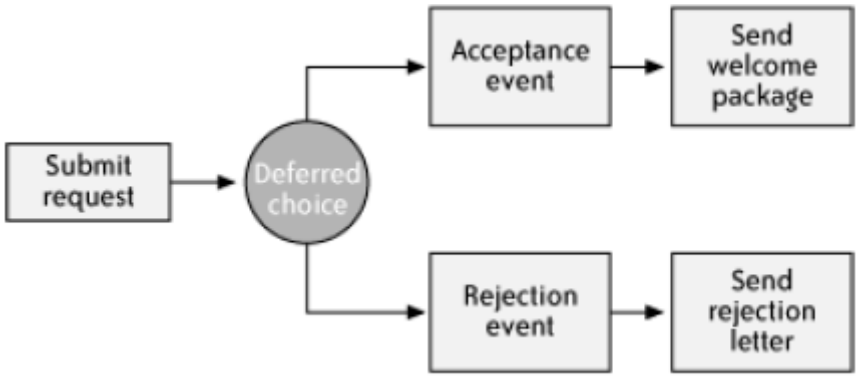


Ilustración 8: Patrón de elección aplazada

En el ejemplo, el proceso está esperando al resultado de la decisión para responder con un evento. Cada uno dirige una ruta de ejecución distinta: el primero lanza un evento para dar la bienvenida y el segundo lanza un evento para enviar una notificación de rechazo.

- Ruta paralela intercalada: consiste en que varias actividades son ejecutadas en orden secuencial (no en paralelo como el nombre del patrón sugiere), pero el orden de

ejecución es arbitrario y no es conocido en tiempo de diseño. Se le conoce también como *Ad Hoc*.

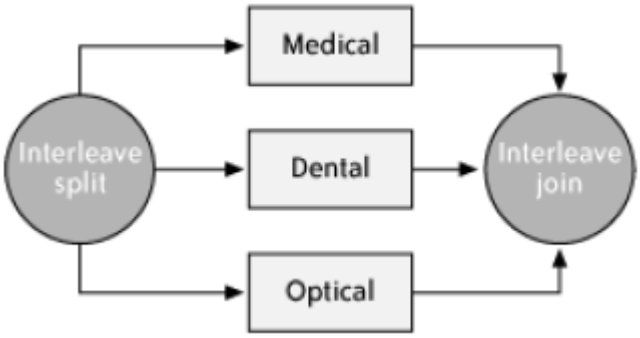


Ilustración 9: Patrón de ruta paralela intercalada

En el ejemplo hay que realizar tres pruebas (óptica, médica y dental), una detrás de otra pero en cualquier orden; cuando una prueba es completada, la siguiente está determinada por la disponibilidad de los médicos. Este mismo proceso podría ser representado con patrones de elección aplazada de la siguiente manera:

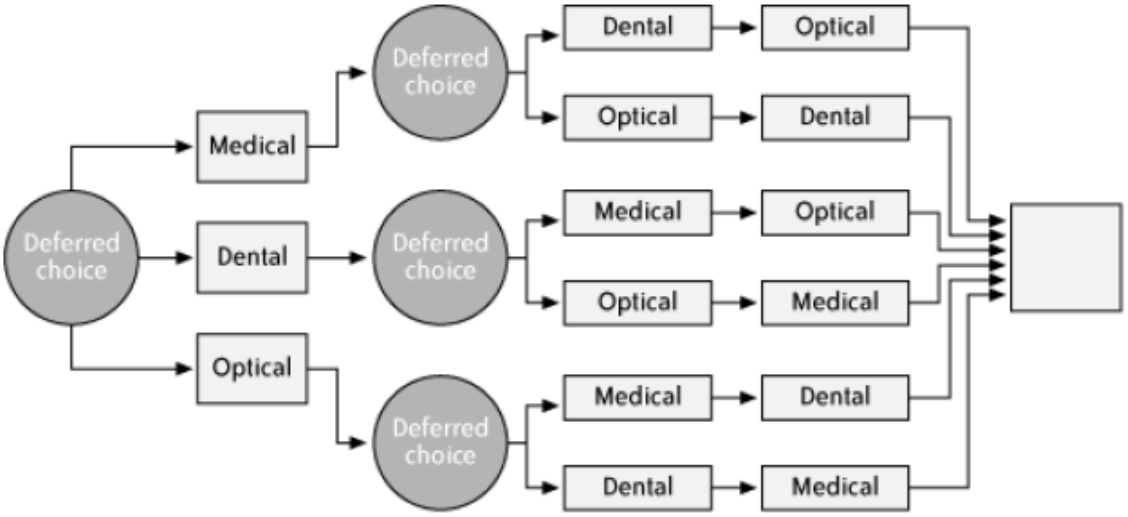


Ilustración 10: Equivalencia a patrón de elección aplazada

- Hito: describe el escenario en el cual una actividad puede ser ejecutada solo después de que la ocurrencia de un evento se permita pero antes de que la ocurrencia del evento se prohíba. Por ejemplo, un comprador puede cancelar un pedido hasta que la orden haya sido enviada.

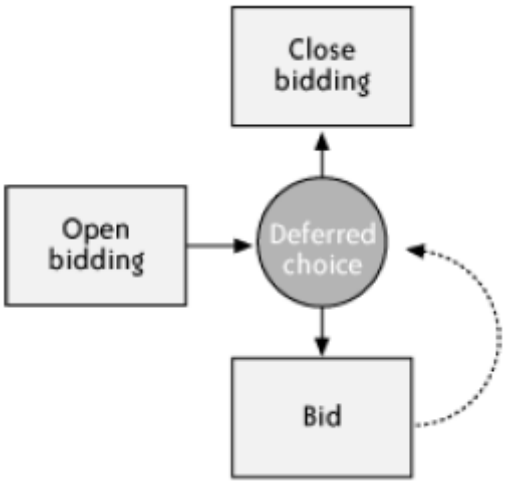


Ilustración 11: Patrón hito

La actividad intermedia Bid ocurre mientras el evento está deshabilitado; cuando el evento se produce e inicia su propia ruta, evita la ejecución de la actividad.

Patrones de cancelación

Si el diseño necesita una estrategia para cancelar el proceso en algún punto durante la ejecución, una forma de resolver el problema puede ser añadir chequeos de cancelación en cada uno de los pasos, pero más deseable es un simple chequeo o acción que cubra la ejecución entera del proceso.

- Cancelar actividad: consiste en parar la ejecución de una particular actividad del proceso con un evento de cancelación. Es también conocido como *matar actividad*. Es usado para abortar una ejecución larga, suspender una actividad o cambiar la ruta del proceso.

Ilustración 12: Patrón de cancelación de actividad

Entre el paso X y el paso Y hay una actividad de larga duración. Para permitir la cancelación de esta actividad durante la ejecución, se establece antes una decisión.

- Cancelación del caso: consiste en parar la ejecución de un proceso entero con un evento de cancelación. Se le conoce también como *matar proceso*.

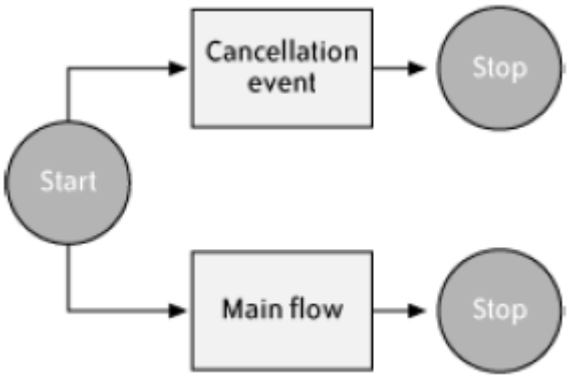


Ilustración 13: Patrón de cancelación del caso

Cuando el proceso comienza, se divide en dos rutas separadas que se ejecutan en paralelo, una con el flujo principal y la otra para la recepción de la cancelación. Si llega un evento de cancelación mientras el flujo principal se está ejecutando, éste termina el proceso entero.

Técnicas de modelado

Las técnicas se refieren a los diagramas y notaciones para desarrollo de sistemas que soportan procesos para analizarlos. En este capítulo se resumen las principales técnicas propuestas para el desarrollo en BPM.

Para decidir qué técnica es la más adecuada para nuestro modelo de proceso es importante tener en cuenta en qué fase del desarrollo se está para buscar una técnica que dé apoyo a alguno de los siguientes modelos:

- *Modelos descriptivos*: dan soporte a las decisiones durante el diseño y desarrollo de los procesos, ayudando a su entendimiento.
- *Modelos de representación*: dan soporte a las decisiones durante la ejecución de los procesos, asegurando un buen rendimiento, controlando y monitorizado los procesos para proporcionar información.

En la elección de la técnica también es importante saber qué perspectiva del modelo es la que buscamos analizar:

- *Perspectiva funcional*: muestra *qué* elementos o actividades constituyen el proceso.
- *Perspectiva de comportamiento*: señala *cuándo* las actividades son ejecutadas, por ejemplo secuencialmente, y *cómo* son ejecutadas, a través de bucles, condiciones para la toma de decisiones, criterios de entrada y salida,…
- *Perspectiva de organización*: indica *dónde* y *por quién* son ejecutadas las actividades, los mecanismos de comunicación usados para transferir datos y los medios físicos usados para almacenarlos.
- *Perspectiva de información*: describe los datos producidos o manipulados por un proceso y sus relaciones.

Diagrama de flujo

Es una representación gráfica de los procesos, centrada en trazar el flujo de información y resaltar las actividades clave y puntos de decisión. La notación facilita la

comunicación con la representación de los procesos a través de gráficos simples. Los símbolos principales del diagrama representan actividades (rectángulos), puntos de decisión (diamantes) y transiciones (flechas).

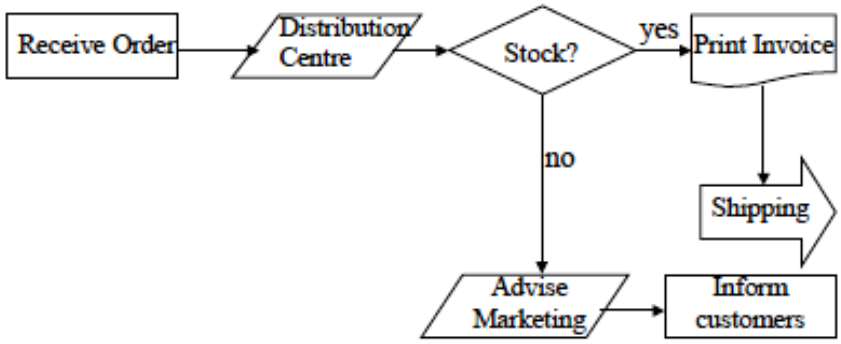


Ilustración 14: Diagrama de flujo

Este proceso comienza cuando un cliente hace un pedido a una compañía. El departamento de ventas recibe el pedido, que lo envía al centro de distribución, el cual verifica la disponibilidad del producto requerido. Si está disponible lo envían al cliente, en otro caso informan al departamento, para que sean ellos los que informen al cliente de la no disponibilidad del producto. Este mismo ejemplo se verá representado desde el punto de vista del resto de técnicas.

Diagrama de Flujo de Datos (DFD)

Es una técnica para la descripción gráfica del flujo de información entre entidades y elementos de almacenamiento de datos en un proceso de negocio. Describe lo que un proceso hace y no cómo lo hace. Son comparables a los diagramas de flujo, aunque los DFD están más orientados a los datos en vez de a las actividades y el control.

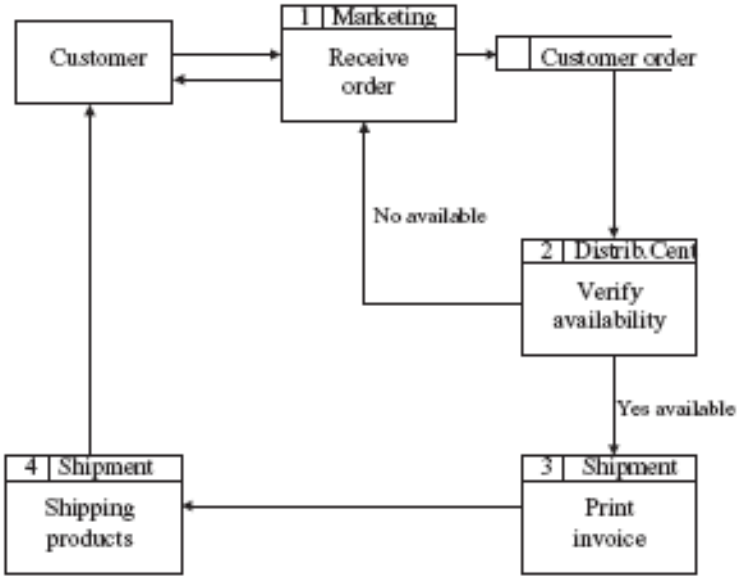


Ilustración 15: Diagrama de flujo de datos

Diagrama de Interacción de Roles (RID)

Está basado en una visión gráfica del proceso desde la perspectiva de los roles, centrándose en las interacciones entre ellos, junto a la lógica que determina cuándo las actividades son llevadas a cabo. Los *roles* son notaciones abstractas que describen un comportamiento deseado dentro de la organización. Es adecuado para el contexto en el cual el elemento humano es un recurso crítico en la organización. En este diagrama las actividades están en el eje vertical de la izquierda y los roles en el eje horizontal superior. Con las líneas horizontales se muestran las interacciones humanas.

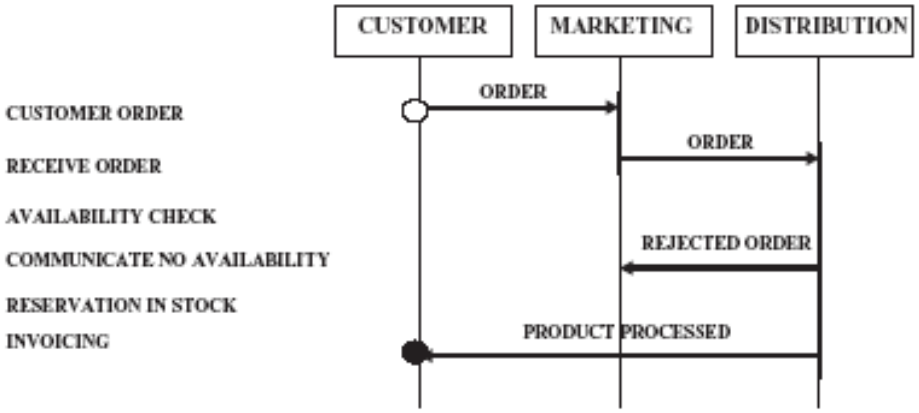


Ilustración 16: Diagrama de Interacción de roles

Diagrama de Gantt

Describe las actividades en una escala de tiempo, para controlar la situación actual de la ejecución. Es una matriz que lista en el eje vertical todas las actividades que son ejecutadas en el proceso. El eje horizontal está encabezado por columnas que indican una estimación de la duración de la actividad en el proyecto, y el nombre de la persona asignada para su realización.

Definición Integrada para Modelado Funcional (IDEF)

Consisten en un número de técnicas independientes desarrolladas como un conjunto de formalismos notacionales, para la representación y el modelado de procesos y estructuras de datos. IDEF0, para el modelado de funciones, e IDEF3, para la captura de procesos, son las técnicas relacionadas principalmente con el modelado de procesos.

IDEF0 muestra las actividades de un proceso a alto nivel, indicando las entradas, las salidas, el control y los mecanismos asociados. Soporta descomposición progresiva en modelos más detallados que describen la jerarquía de descomposición de las actividades. Muestra qué se hace en la organización, por lo que se dirige hacia la perspectiva funcional.

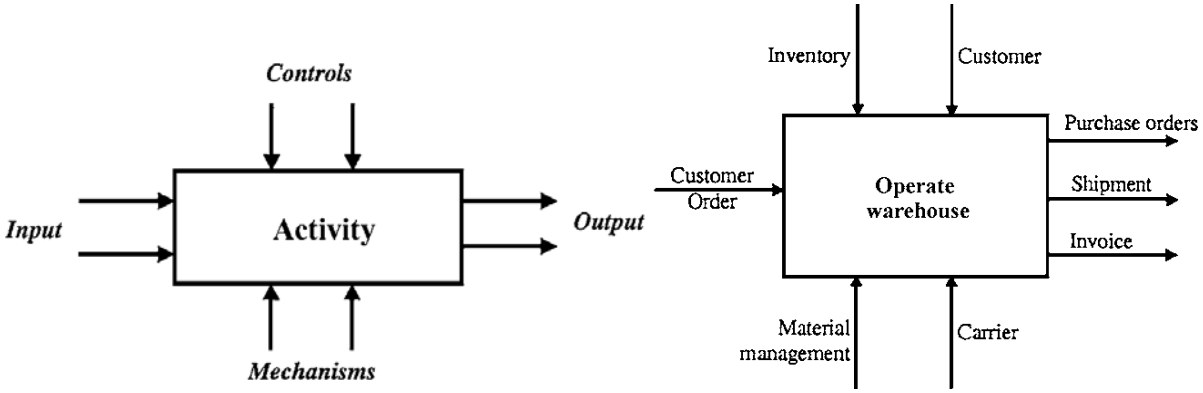


Ilustración 17: IDEF0

IDEF3 es el método de descripción usado para capturar los aspectos de comportamiento de un proceso. Las descripciones muestran las relaciones entre las actividades y los eventos de los procesos. Es una técnica de modelado basada en la captura de relaciones de precedencia y causalidad entre situaciones y eventos. Usa dos representaciones gráficas: el *diagrama de flujo del proceso*, el cual describe el flujo entre actividades, y el *diagrama de transición de estados*, que representa los diferentes estados por los que

pasan las entidades a través del proceso, suministrando información sobre secuencias relacionadas con el tiempo. La notación básica del método consiste en una serie de rectángulos que representan las actividades, círculos que representan estados y arcos que enlazan entre ellos, que representan transiciones o cambios de estado. Muestra cómo se trabaja en la organización, al contrario que un IDEF0, el cual se encarga de qué actividades ejecuta la organización.

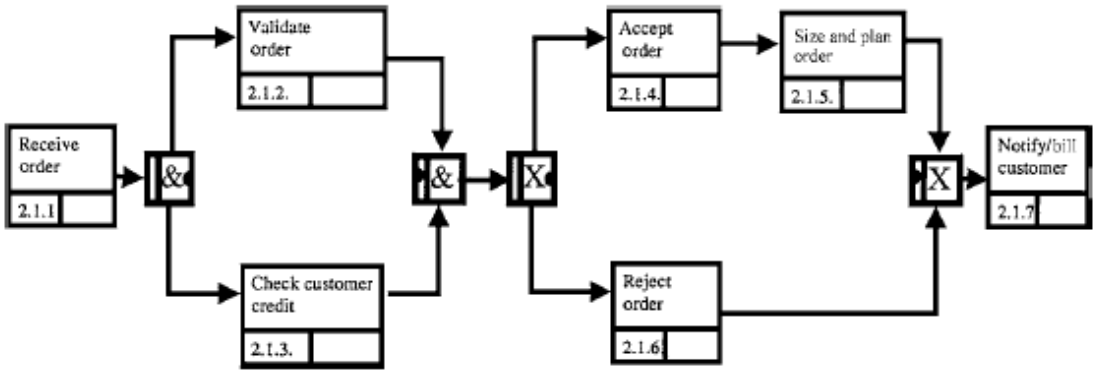


Ilustración 18: IDEF3 (Diagrama de flujo)

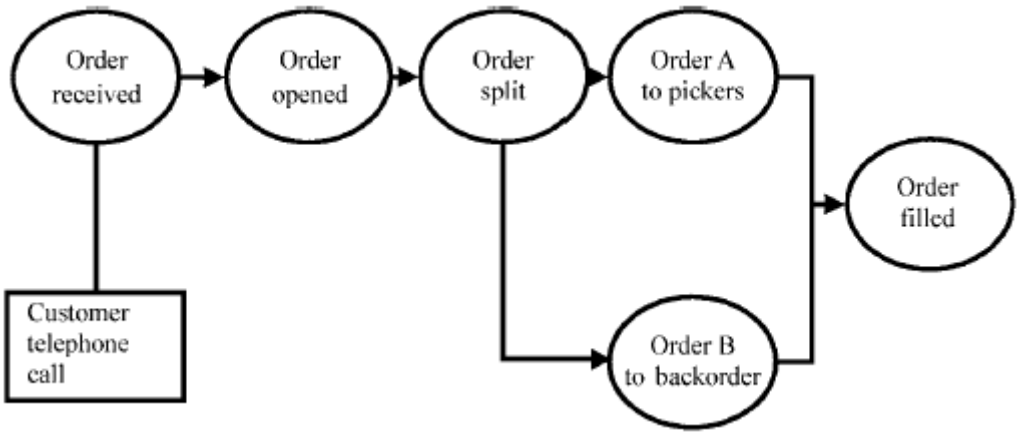


Ilustración 19: IDEF3 (Diagrama de transición de estados)

Redes de Petri

Las redes de Petri son un lenguaje gráfico basado en estados para el modelado de procesos que ayuda a describir la semántica del flujo de control. Es usado en el análisis del comportamiento dinámico del sistema, para permitir comprender cómo interactúan los procesos.

Los símbolos que utiliza para la representación son:

- *Lugar*: se dibuja con un círculo, y es un punto de parada en el proceso.
- *Transición*: es un rectángulo que representa un evento o acción.
- *Ficha*: es un punto negro que está en un lugar. El conjunto de fichas representan el estado actual del proceso, permitiendo ver la concurrencia en los lugares. Durante la ejecución de los procesos, las fichas se mueven de lugar en lugar.
- *Arco*: es un enlace de una transición a un lugar o de un lugar a una transición.

Hay algunas reglas elementales a la hora de realizar un diseño:

- Un arco puede conectar un lugar a una transición, o una transición a un lugar, pero nunca un lugar a un lugar, o una transición a una transición. Así, el orden de los lugares y transiciones alterna en una ruta:
  lugar → transición → lugar → transición → lugar
- Una ruta debe comenzar y terminar con un lugar.
- La relación entre lugar y transición va de varios a varios. Es decir, un lugar puede ramificarse en múltiples transiciones y varios lugares pueden unirse en una sola transición; de la misma forma, una transición puede ramificarse en múltiples lugares y varias transiciones pueden converger en un solo lugar.
- Las fichas encajan en los lugares. Una transición no puede contener una ficha.

Al simular un diseño se inicia la ejecución de la red de Petri, en la cual cada paso dispara una transición y muestra las fichas avanzadas a los nuevos lugares. Las *reglas de ejecución* son:

- Una transición está permitida si cada lugar de entrada que enlaza a ella, tienen al menos una ficha.
- Cuando una transición dispara, consume una ficha de cada lugar de entrada y genera una ficha por cada lugar de salida con el que enlaza.

El siguiente ejemplo es un proceso en el cual los editores de una revista deciden si aceptar un artículo para su publicación. Cuando el artículo llega, es distribuido entre los editores para su revisión. La votación se completa cuando dos editores lo aceptan o

cuando nadie lo rechaza, en cualquier caso al autor se la envía una notificación con el resultado.

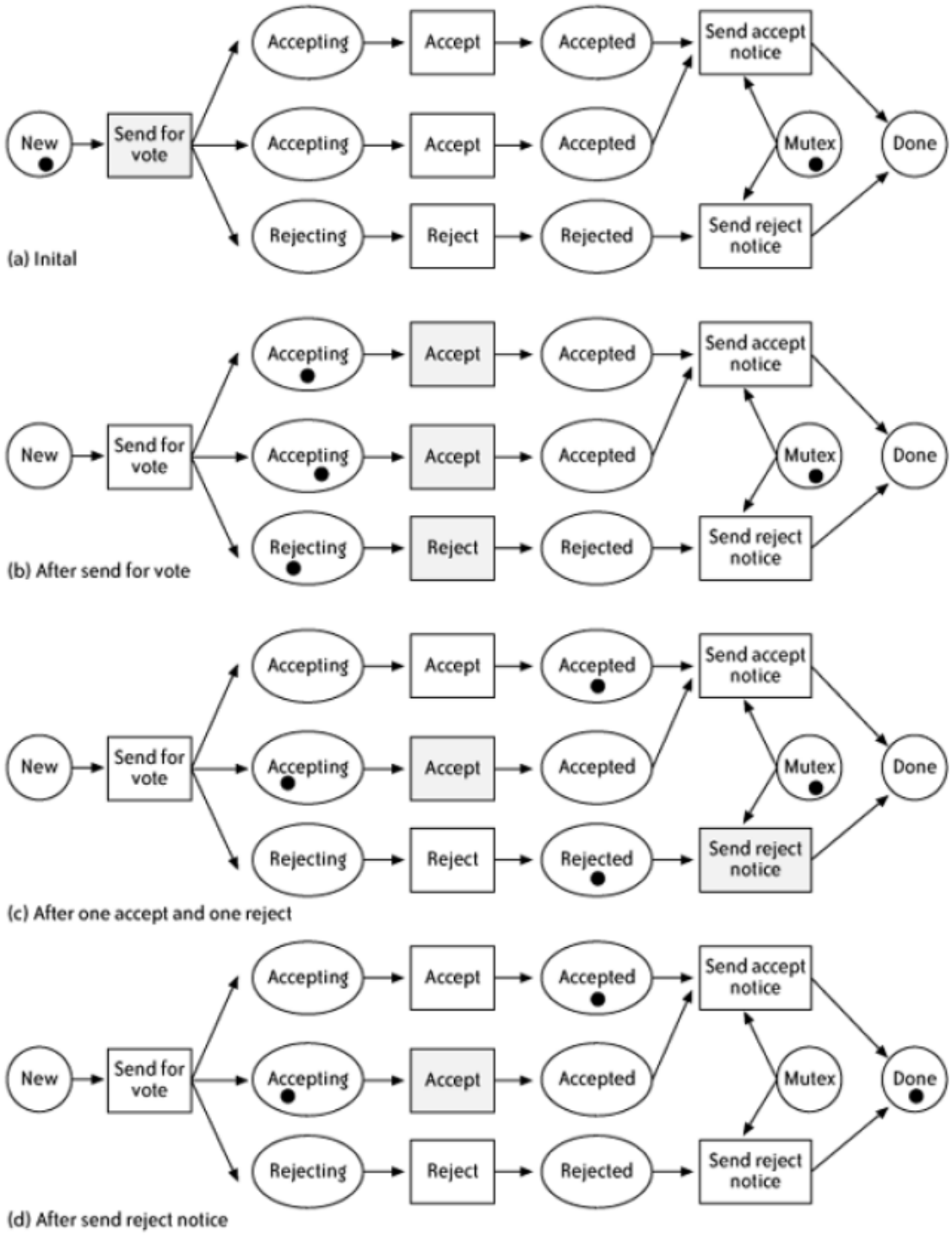


Ilustración 20: Red de Petri

Inicialmente las fichas están colocadas en los lugares *New* y *Mutex*, y la transición *Send for vote* está permitida, porque su única entrada tiene una ficha. Cuando *Send for vote* se dispara, la ficha es eliminada de *New* y llega a los lugares *Reject* y *Accepting*, permitiendo las transiciones *Accept* y *Reject*. En el siguiente paso se describe el estado de la red después de una aceptación y un rechazo, lo que deja a la ficha en los lugares *Accepted* y *Rejecting*. De las dos transiciones para el envío de noticias, ahora solo *Send reject notice* esta permitida, porque *Reject* y *Mutex* tienen ficha. Por último, cuando la noticia de rechazo es enviada, la ficha de *Reject* y *Mutex* es movida hasta *Done*.

La transición *Mutex* asegura que solo una noticia, la de aceptación o la de rechazo, va a ser enviada al autor, debido a que inicialmente solo tiene una ficha, y en el caso de que ocurran dos aceptaciones y un rechazo al mismo tiempo, las transiciones *Send reject notice* y *Send accept notice* compiten por la misma ficha, y cuando una transición la dispare, consumirá la ficha, lo que evita que sea disparada por la otra.

Simulación

Su objetivo es adquirir conocimiento para tomar una decisión con respecto a un sistema del mundo real, pero el sistema no es fácil de estudiar directamente debido a restricciones de coste, de tiempo o humanas. Por tanto se procede indirectamente a crear un modelo simulado, que imita el comportamiento real y del que se está seguro que algo de lo que aprendamos sobre el modelo será verdad en el sistema, con el propósito de evaluar estrategias y de comprender el comportamiento real.

Técnicas basadas en el conocimiento

Los sistemas basados en el conocimiento utilizan técnicas de inteligencia artificial que explotan el uso del conocimiento sobre una organización para soportar el razonamiento inteligente sobre un modelo, donde se describen los posibles comportamientos para el proceso modelado.

Estándares para modelar

Los estándares sirven para la adopción de un vocabulario común que describa los procesos de negocio. A la hora de elegir los criterios para conseguir un estándar de modelado completo y unificado se han de valorar los siguientes aspectos:

- *Metamodelo*: define el lenguaje o conjunto de conceptos y relaciones con el que se construye la representación abstracta del modelo de proceso. Debe estar definido estricta y consistentemente para suministrar un fundamento sólido.

- *Representación en serie*: el modelo generado debe ser guardado en cierto formato para el intercambio entre diferentes herramientas, es lo que se conoce como representación en serie. La mayoría de los modelos adopta XML como formato.

- *Extensible*: aunque los modelos están diseñados para contener a la mayoría de construcciones, puede haber circunstancias bajo las cuales se necesita información adicional. Por lo que la habilidad de extender o personalizar el modelo es bastante importante.

- *Soporte en herramientas*: si un estándar de modelado no está soportado por ninguna herramienta, tiene obstáculos para ser usado en la práctica. Algunas herramientas son de propósito general y pueden ser usadas por varios modelos; también las hay de propósito especial, que solo pueden ser usadas por modelos específicos.

Entre los estándares existentes, se ha elegido la propuesta de arquitectura BPM inspirada en el modelo de referencia de la Workflow Management Coalition (WfMC), que propone Business Process Execution Language (BPEL), Business Process Modeling Notation (BPMN) y Web Services Choreography Description Language (WS-CDL) como estándares de ejecución, diseño y coreografía, respectivamente, para las partes más importantes de BPM. El corazón de esta arquitectura es un motor que ejecuta procesos cuyo código fuente está escrito en lenguaje BPEL. Estos procesos están diseñados por analistas que usan un editor gráfico que soporta el lenguaje visual de diagramas de flujo en BPMN. El editor incluye la exportación a código BPEL a partir de los diagramas BPMN.

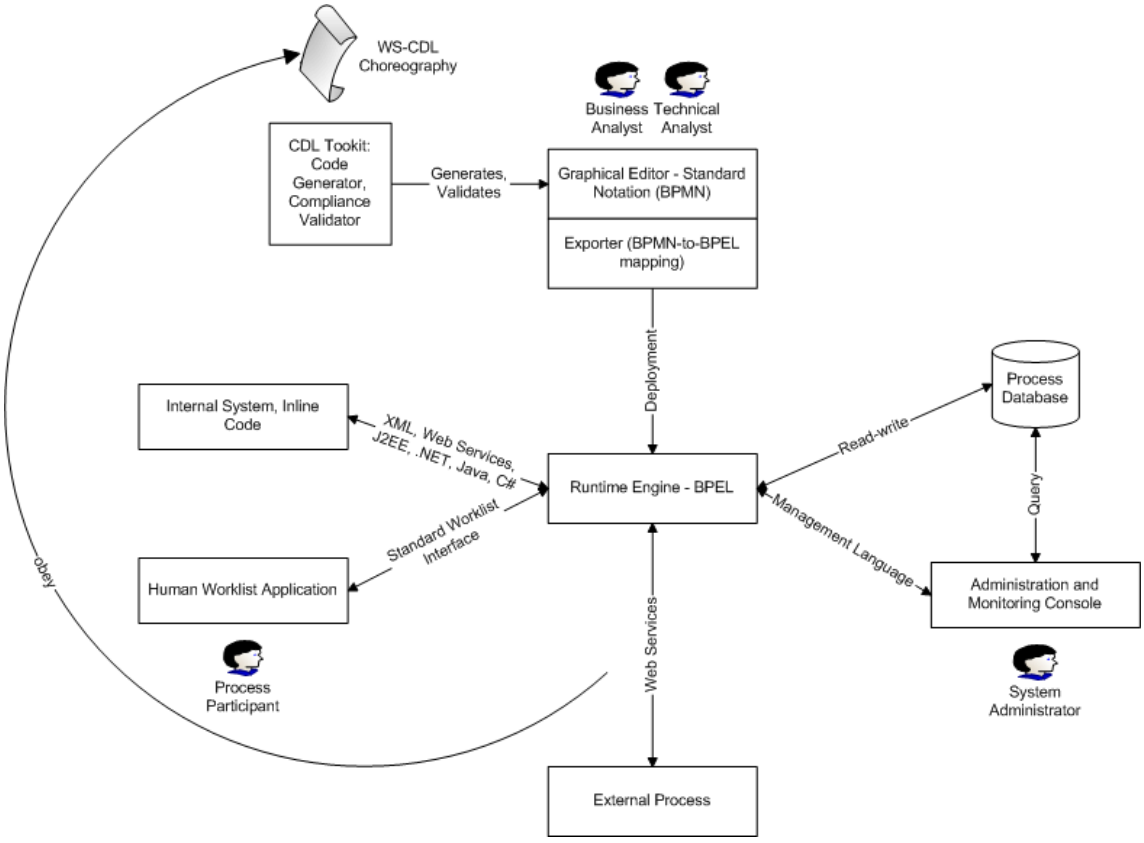


Ilustración 21: Modelo de referencia WfMC

Workflow Management Coalition (WfMC)

Es un modelo de referencia que describe a alto nivel las principales piezas de una arquitectura BPM. Sus componentes son:

- *Herramienta de definición de procesos*: es un editor usado para componer una definición de proceso de negocio. Normalmente es una herramienta gráfica que permite al usuario construir el proceso dibujándolo respecto a la notación elegida. El requisito clave está en la habilidad de exportar la definición del proceso a un formato estándar, el cual pueda ser ejecutado por un servicio de representación.

- *Servicio de representación de flujos de trabajo*: acepta como entrada definiciones de procesos importadas desde la herramienta de definición. El servicio administra la ejecución de las instancias de estos procesos. También

suministra interfaces a aplicaciones internas o externas y herramientas para la administración y monitorización que leen el estado de los procesos.

- *Aplicaciones cliente de flujos de trabajo*: suministran una interfaz para que las personas participen en el proceso. Clasifica los trabajos para recordar las actividades que deben ser completadas por una persona.

- *Aplicaciones invocadas*: son aplicaciones externas llamadas por el proceso desde una actividad, usando mecanismos de comunicación con otros sistemas, como los servicios web.

- *Herramientas de administración y monitorización*: incluye la administración de usuario y roles, y la monitorización de las instancias, junto con la configuración del estado y la terminación de los procesos bloqueados.

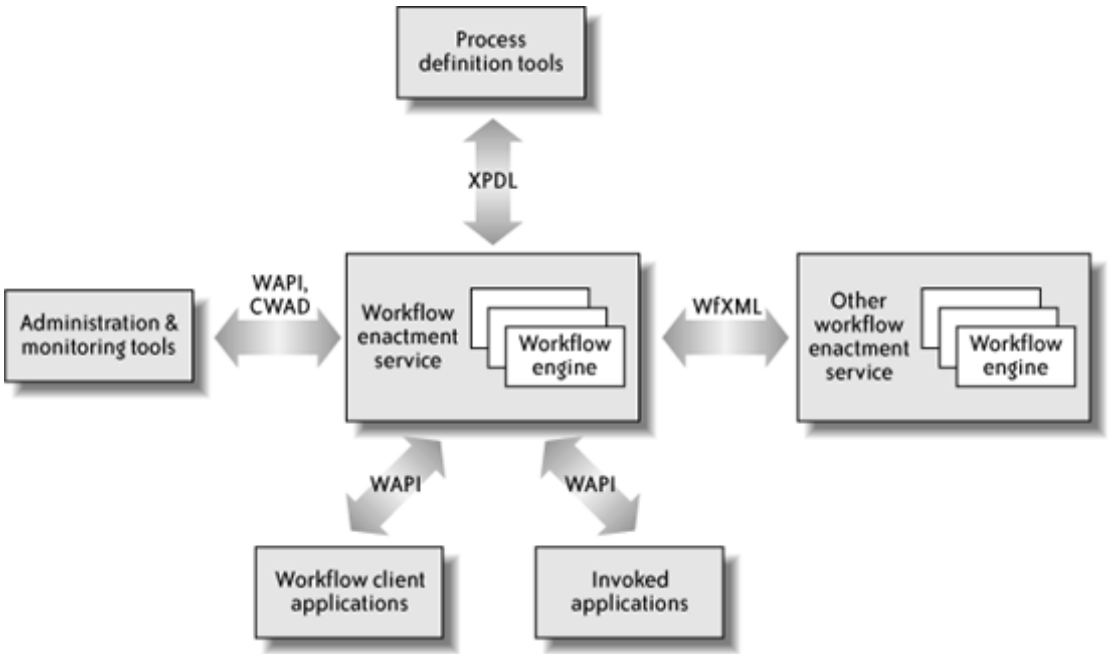


Ilustración 22: Workflow Management Coalition

Business Process Execution Language (BPEL)

Es el lenguaje estándar para la ejecución de definiciones de procesos de negocio basado en servicios web como mecanismo de comunicación para interactuar con entidades externas.

Un documento BPEL representa la lógica del proceso asociada y los servicios web con los que se verá relacionado. Si imaginamos un flujo de negocio con una entrada y una salida, entremedias el flujo se compone de muchos procesos internos que se lanzan dependiendo de los eventos que ocurran y BPEL es el encargado de la integración de todos estos procesos automáticos, ordenando qué proceso ejecutar y en qué momento.

Otra característica importante es su lenguaje expresado en sintaxis XML que facilita el formato de intercambio para soportar la transferencia de definiciones de procesos entre herramientas.

Web Services Choreography Description Language (WS-CDL)

WS-CDL establece un lenguaje basado en XML, que suministra una forma estándar de especificar la captura, desde el punto de vista global, del comportamiento común de interacción entre participantes, a través de las reglas de intercambio de mensajes. Las principales características son:

- En la coreografía los participantes cooperan a través del intercambio de mensajes, *sin depender de una máquina de control centralizado* que coordine las interacciones. De hecho, cada participante tiene su propia máquina que ejecuta sus procesos de negocio, y que son los que cumplen las reglas de interacción requeridas por la especificación. Es decir, un proceso conoce por la coreografía cómo llamar a los servicios que necesita, y cómo pueden llamar a sus servicios. Esta información permite generar el esqueleto de código de procesos para la ejecución, con actividades que hagan de punto de encuentro para estos servicios. De esta forma complementa a los lenguajes de proceso, proponiendo un modelo de interacción global.

- Rara vez se usa en aplicaciones de intranet, normalmente los participantes pertenecen a diferentes *compañías globalmente distribuidas*, que usan los servicios para comunicarse con otros participantes, y que a su vez publican sus interfaces de servicios (*solo el comportamiento público es relevante*).

Lenguaje Unificado de Modelado (UML)

Antes de ver en detalle el estándar BPMN en el siguiente capitulo, resumimos las características de otro estándar para el diseño por su gran aceptación en este ámbito. UML es considerado el lenguaje gráfico orientado a objetos estándar. Representa a la mayoría de las técnicas usadas con un conjunto de construcciones de modelado y con una notación común. Consiste en nueve diferentes tipos de diagramas, donde cada uno muestra un aspecto del sistema:

- *Diagrama de clases*: describe la estructura del sistema con clases y relaciones. Captura el vocabulario del sistema. El ejemplo del proyecto incluye un diagrama de clases como parte del análisis, véase la página 54.
- *Diagrama de objetos*: muestra combinaciones de objetos y sus relaciones en un instante de tiempo.
- *Diagrama de estados*: expresa los posibles estados por los que pasa un objeto en respuesta a eventos. Se usa cuando el comportamiento de un objeto varía dependiendo de su estado actual.
- *Diagrama de actividades*: describe las actividades y acciones que ocurren en un sistema. Es el más utilizado para modelar los procesos de negocio. Se puede ver ejemplos de estos diagramas en el apartado de diseño del ejemplo realizado para el proyecto de la página 60 a la 65.
- *Diagrama de secuencia*: muestra secuencias de mensajes enviados entre un conjunto de objetos.
- *Diagrama de colaboración*: describe una colaboración entre un conjunto de objetos.
- *Diagrama de casos de uso*: ilustra las relaciones entre casos de uso, capturando la funcionalidad del sistema desde el punto de vista del usuario. Estos diagramas también han sido utilizados en el ejemplo del proyecto entre la página 55 y 59.
- *Diagrama de componentes*: describe los componentes software del sistema.
- *Diagrama de despliegue*: describe el hardware del sistema.

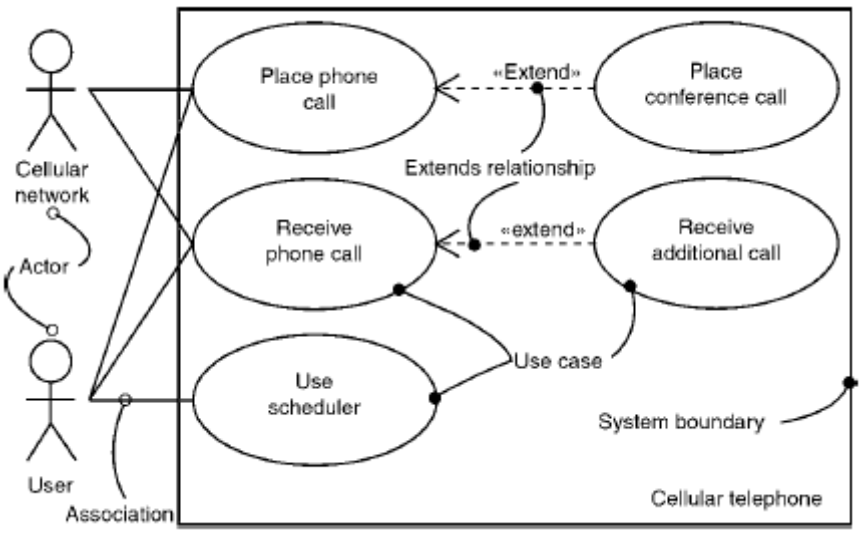


Ilustración 23: Diagrama de casos de uso

Business Process Modeling Notation (BPMN)

La Notación para el Modelado de Procesos de Negocio (BPMN) es un estándar basado en los diagramas de flujo, adaptado para suministrar una notación gráfica que posibilita la representación de procesos de negocio a través de flujos de trabajo. Su objetivo principal es el de coordinar la secuencia de actividades involucradas en el proceso. Fue diseñado para facilitar la lectura y el uso intuitivo con una notación comprensible para todos los usuarios, desde los analistas de negocio que crean el borrador inicial de los procesos, a los desarrolladores técnicos responsables de la implementación de la tecnología que ejecuta esos procesos, y finalmente la gente del negocio la cual administra y monitoriza los procesos.

Hasta ahora el diseño de los modelos de procesos de negocio está separado de la representación requerida para la implementación y ejecución de los procesos. Por lo que hay una necesidad de traducir manualmente el modelo original al modelo de ejecución, estando esta acción sujeta a errores. Es por tanto un objetivo clave crear un puente desde la notación del modelado de procesos a los lenguajes de ejecución que implementen los procesos. En este aspecto BPMN define la especificación de las reglas que permiten la generación de un ejecutable BPEL a través del mapeo entre los objetos gráficos de BPMN y el código BPEL.

Dependiendo del propósito que se tenga para el proceso de negocio hay tres modelos básicos que pueden ser creados con BPMN:

- *Procesos globales o de colaboración entre negocios*: muestra las interacciones entre dos o más entidades a través del intercambio de mensajes desde un punto de vista global, es decir, no se toma la visión de un participante particular. Es la parte coreográfica del lenguaje.

- *Procesos privados de negocio*: enfocados a actividades internas de una organización, no son visibles al público.

- *Procesos abstractos de negocio*: representan las actividades públicas usadas por los procesos para comunicarse con otros.

Notación para modelos en BPMN

Para empezar a crear un modelo de proceso de negocio hay que saber que el diseño debe cumplir con las siguientes condiciones:

- Permite múltiples eventos de inicio y fin, pero debe haber al menos uno de inicio y otro de fin; los eventos de inicio no tienen entradas, pero sí una única salida, mientras que los eventos de fin tienen una sola entrada y ninguna salida.
- Cada actividad tiene exactamente una entrada y una salida.
- Toda actividad está en una ruta que va desde un evento inicio a un evento fin.

Estos elementos y algunos más están clasificados en cuatro categorías que contienen a los elementos básicos para el diseño de modelos en BPMN y que se explican detalladamente a continuación:

- *Objetos de flujo*: eventos, actividades, puntos de decisión.
- *Objetos de conexión*: flujo secuencial, flujo de mensaje, asociación.
- *Calles de piscina*: piscina, calle.
- *Artefactos*: objetos de datos, grupos, anotaciones de texto.

Objetos de flujo

Son los elementos gráficos que definen el comportamiento de un proceso de negocio.

Eventos

Un evento es algo que ocurre durante un proceso. Los eventos afectan al flujo del proceso y tienen una causa y un resultado. Están categorizados por la etapa en la cual ocurre en el proceso: inicio, intermedio y fin, y por el tipo: básico, mensaje, tiempo, regla, excepción, cancelación, compensación, enlace, múltiple o terminación. La forma de un evento es un círculo pequeño: un evento de inicio tiene un borde fino, un evento de fin tiene un borde grueso y un evento intermedio tiene un borde doble. En la siguiente ilustración se muestra qué símbolo corresponde a cada tipo de evento y el significado de cada uno se indica en la tabla que va a continuación.

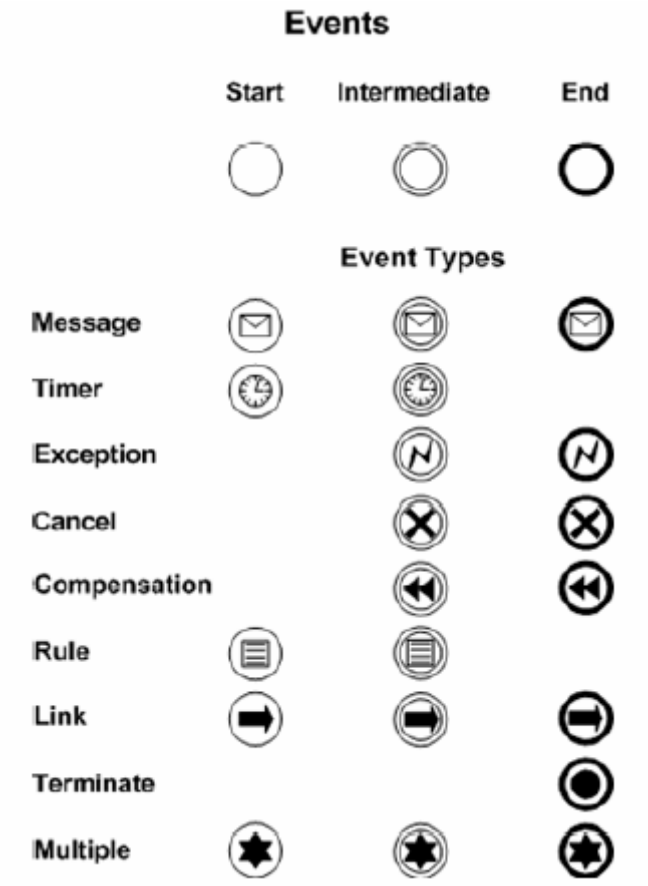


Ilustración 24: Eventos BPMN

Tipo	Inicio	Intermedio	Fin
Mensaje	Recepción de un mensaje	Espera de un mensaje	Envío de un mensaje
Tiempo	Define una tarea programada	Programación rechazada	
Excepción		Lanza o captura un error	Genera un error
Cancelación		Ejecuta la cancelación de una actividad	Cancela la transacción
Compensación		Lanza el manejo de compensaciones	Ejecuta la compensación
Regla	Condición satisfecha	Condición satisfecha para iniciar el manejo de excepciones	
Enlace	Se conecta al final de un proceso	Enlaza desde o hacia otra actividad	Conecta con el inicio de un proceso
Terminación			Termina todas las actividades del proceso. No ejecuta manejo de excepciones ni de compensación

Múltiple	Dos o más triggers pueden iniciar el proceso; es suficiente con que uno de ellos ocurra	Dos o más triggers pueden reanudar un proceso en espera; es suficiente con que uno de ellos ocurra	Varios resultados son requeridos para acabar el proceso

Tabla 1: Tipos de eventos BPMN

Actividades

Las actividades son acciones ejecutadas durante el proceso. Se representan con rectángulos redondeados. Los modelos pueden contener los siguientes tipos de actividad dependiendo de si es atómica o compuesta:

- *Tarea*: son actividades que ejecutan una simple acción. Representan el nivel más bajo del proceso, por lo que no pueden ser descompuestas en subtareas.

Ilustración 25: Tareas BPMN

- *Subproceso*: es un proceso compuesto por actividades. Permite el desarrollo jerárquico de los modelos de procesos. Puede ser descompuesta con mayor nivel de detalle a través de subactividades. Hay de dos tipos:

  - Subproceso expandido: muestra los detalles de un subproceso.
  - Subproceso contraído: muestra el elemento de nivel superior, donde los detalles del subproceso no son visibles. Un icono con un + en la parte inferior del elemento indica que existe una capa adicional de mayor complejidad.

Ilustración 26: Subprocesos BPMN

Como se puede ver en las ilustraciones anteriores las actividades además son marcadas con símbolos que representan al tipo de acción que realizan:

- *Bucle y múltiples instancias*: son actividades que pueden ser configuradas para iterar en modo estándar (while o until) o para soportar la ejecución de múltiples instancias en paralelo (foreach). Una actividad en modo estándar se indica con una flecha circular y una actividad con múltiples instancias tiene un símbolo con un par de barras paralelas.

- *Ad hoc*: contiene un conjunto de actividades que ocurren en cualquier orden, por lo que el flujo de control está sin estructurar.

- *Compensación*: es una actividad fuera del flujo normal del proceso, capaz de revertir la lógica de una actividad, deshaciendo sus efectos, después de que se hayan completado. Se indica con un símbolo de rebobinar.

Punto de decisión

Son usados para modelar decisiones en el flujo de secuencia. Tiene dos modos de uso: puede representar la división de una ruta de entrada en varias rutas de salida (modo división) o la unión de varias rutas de entrada en una ruta de salida (modo unión). Su símbolo es un diamante. Varios puntos de decisión pueden ser aplicados:

- *Incluido*: en modo división, permite continuar a las rutas de salida cuya condición se evalúa a cierto. En modo unión, bloquea el paso hasta que las rutas de entrada en ejecución han sido completadas.
- *Paralelo*: en modo división, permite continuar en paralelo a todas las rutas de salida. En modo unión, bloquea hasta que todas las rutas de entrada han sido completadas.
- *Exclusivo para datos*: en modo división, evalúa una condición de cada ruta de salida y permite la primera ruta cuya condición se evalúa a cierto; las demás son ignoradas. Una ruta por defecto puede ser especificada en caso de que ninguna de las otras rutas sea lanzada. En el modo unión, permite continuar a la primera de las múltiples rutas de entrada y todas las demás son descartadas.
- *Exclusivo para eventos*: en modo división, cada ruta lleva a un evento y permite continuar al primer evento lanzado.
- *Complejo*: en modo división o unión evalúa una expresión para determinar a cuál de las rutas de salida o entrada permite continuar.

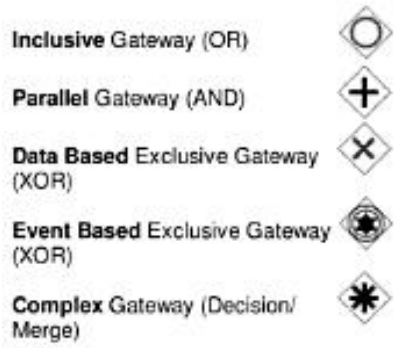


Ilustración 27: Puntos de decisión BPMN

Objetos de conexión

Conectan los elementos que forman el diagrama mostrando el flujo de control del proceso.

Flujo

- *Flujo secuencial*: conecta actividades, eventos y decisiones para mostrar el orden en el que son ejecutados en el proceso. Está representado por una línea sólida con una flecha compacta en la punta.

Ilustración 28: Flujo secuencial en BPMN

- *Flujo condicional*: tiene una expresión condicional la cual es evaluada para determinar si se continúa el flujo o no. Para representarlo un diamante pequeño se pone al principio de la flecha.

Ilustración 29: Flujo condicional en BPMN

- *Flujo por defecto*: es usado si todos los otros flujos condicionales son falsos en un punto de decisión. Una barra en diagonal al inicio de la flecha es usada como indicación visual.

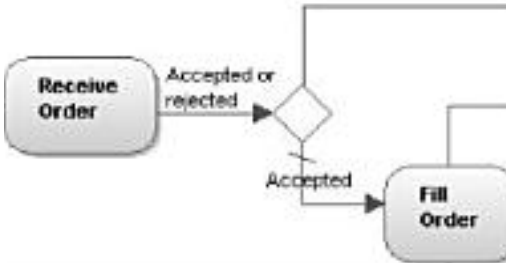


Ilustración 30: Flujo por defecto en BPMN

- *Flujo de mensaje*: muestra el flujo de mensajes enviados y recibidos entre participantes separados por piscinas en el diagrama. Está representado por una línea punteada con una flecha hueca en la punta. Más adelante se detalla el uso de las piscinas y se incluye un ejemplo donde se puede ver este tipo de flujo.

Asociación

Son usadas para asociar datos, texto u otros artefactos sin flujo con objetos del flujo. Está representado por una línea a rayas con una flecha simple en la punta. Más adelante

se describen los tipos de artefactos permitidos y se pueden ver su uso a través de las asociaciones.

Calle de piscina

Organiza las actividades en categorías separadas visualmente para ilustrar diferentes capacidades funcionales o responsabilidades.

Piscina

Es para procesos de negocio que muestran las interacciones entre varios participantes (entidades o roles). Cada participante es representado por un rectángulo llamado piscina, que divide y organiza sus actividades en la colaboración. El flujo de mensajes muestra la comunicación entre los participantes. El flujo de secuencia no puede cruzar los límites de la piscina.

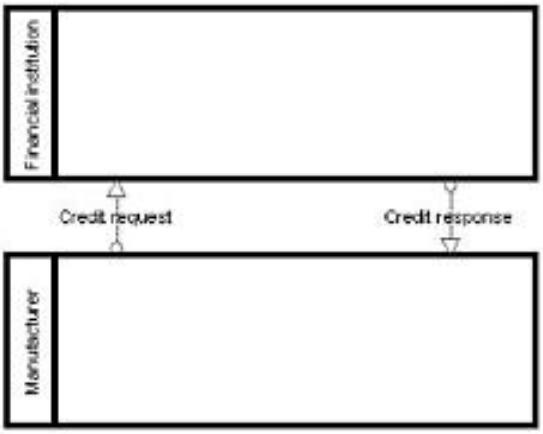


Ilustración 31: Piscinas en BPMN

Calle

En un proceso que abarca múltiples participantes, las piscinas pueden ser subdivididas en calles, las cuales clasifican las actividades de la piscina, describiendo quién hace qué y cómo se estructuran las interacciones. El flujo de secuencia puede cruzar los límites de las calles dentro de la piscina, pero el flujo de mensaje no puede ser usado entre los objetos de las calles de una misma piscina.

Artefactos

Los artefactos permiten mostrar información adicional para una situación específica de modelado, indicando los datos, documentos y otros objetos que son usados. Son conectadas a los elementos del flujo usando una asociación.

Objetos de datos

Son documentos que muestran qué datos son requeridos o producidos por las actividades durante el proceso.

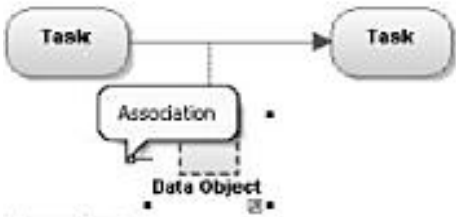


Ilustración 32: Objeto de datos asociado en BPMN

Anotación de texto

Permiten incluir información de texto adicional en un proceso.

Grupo

Los grupos resaltan secciones de un diagrama, incluso a través de diferentes piscinas. Es usado para propósitos de documentación o análisis, sin afectar al flujo de secuencia o de mensaje. Está representado por un rectángulo de esquina redondeada dibujado con línea discontinua.

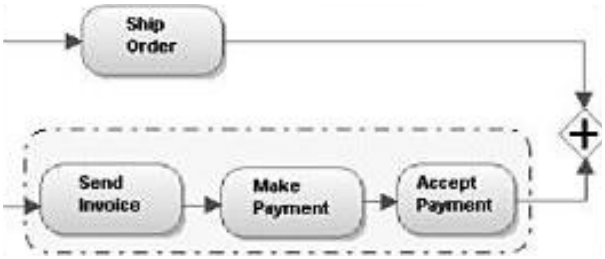
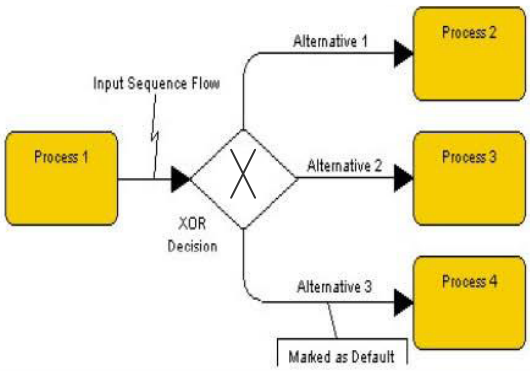


Ilustración 33: Grupo en BPMN

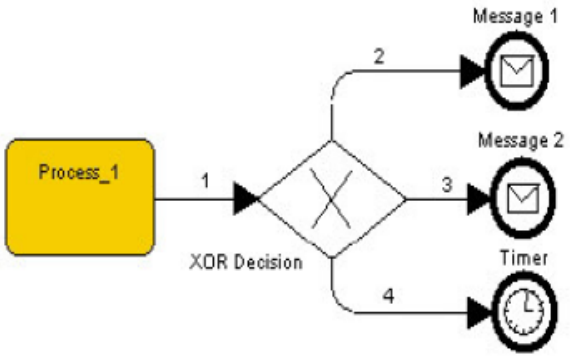
Patrones en BPMN

Los patrones de diseño son representados con BPMN a través del control del flujo con puntos de decisión.

- *Patrón de elección exclusiva*: representa un punto en el proceso donde la decisión se toma por elección de una de las posibles rutas. Está marcada por un punto de decisión exclusivo basado en datos o en eventos.



Patrón de elección exclusiva basado en datos



Patrón de elección exclusiva basado en eventos

- *Patrón de unión exclusiva:* representa un punto en donde dos o más bifurcaciones son unidas sin sincronización y donde ninguna de estas bifurcaciones ha sido ejecutada en paralelo con alguna otra. Usa un punto de decisión exclusivo basado en datos.

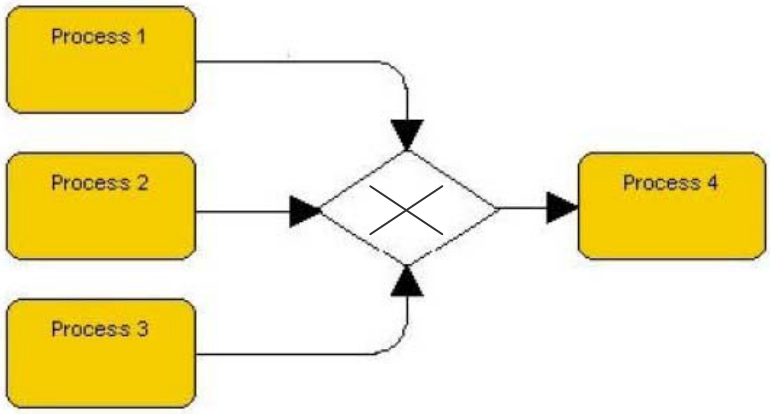


Ilustración 34: Patrón de unión exclusiva en BPMN

El uso de estos dos patrones puede verse en los diagramas realizados para el ejemplo del proyecto, en concreto es utilizado para el alta de una revista, donde después de que los datos de entrada sean evaluados, se decide si se rechaza la revista porque no son correctos o se graban en la base de datos, véase en la página 61.

- *Patrón de división paralela:* marca el punto a partir del cual todas las rutas de salida son continuadas en paralelo. Utiliza un punto de división paralelo en modo división.

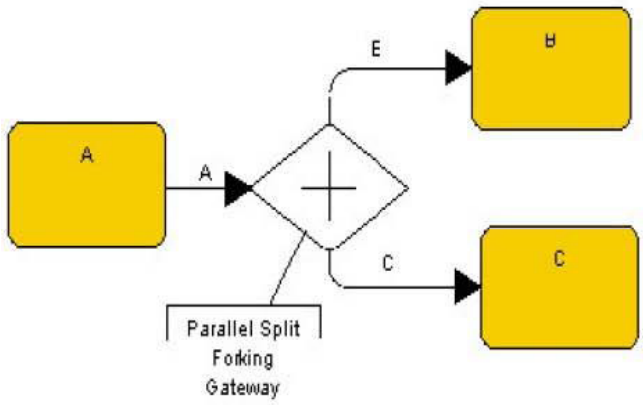


Ilustración 35: Patrón de bifurcación paralela en BPMN

- *Patrón de unión paralela:* espera a que todas las señales de entrada lleguen al punto de decisión para obtener una respuesta de salida. Utiliza el punto de decisión paralelo en modo unión.

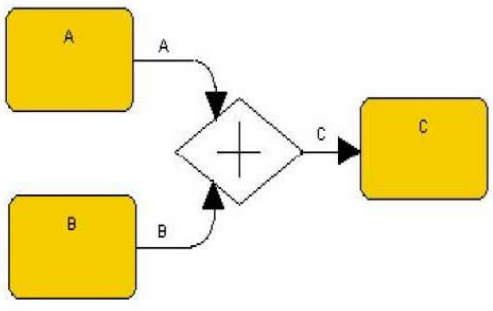


Ilustración 36: Patrón de unión paralela en BPMN

Estos dos últimos patrones también han sido aplicados en el proyecto, por ejemplo en el diagrama de alta de una revista, donde las comprobaciones que componen la validación de los datos de entrada podrían hacerse a la misma vez, esto se puede ver en la página 61.

- *Patrón de elección múltiple:* marca un punto donde después de considerar una cierta decisión, un número de bifurcaciones puede ser seleccionado. Usa un punto de decisión incluido en modo división.

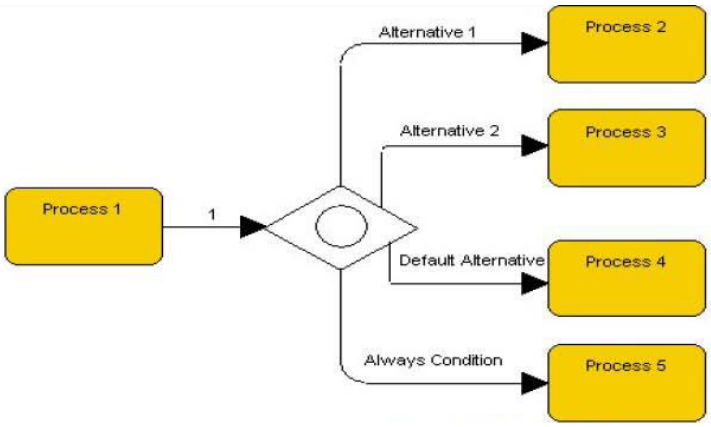


Ilustración 37: Patrón de elección múltiple en BPMN

En evaluaciones donde la decisión requiere de una condición de mayor expresión se utiliza puntos de decisión complejos.

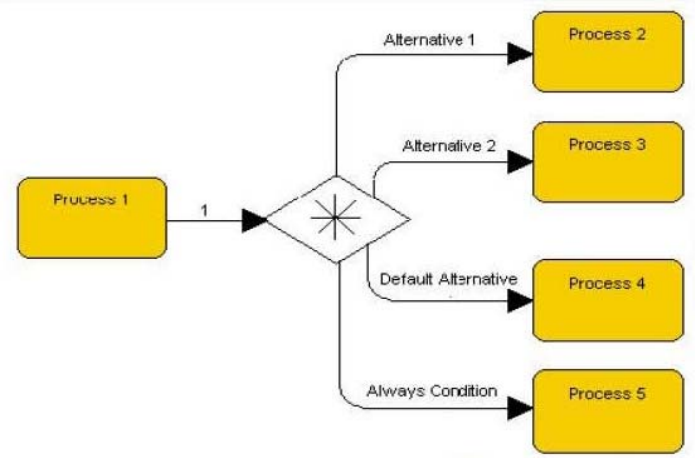


Ilustración 38: Patrón de elección múltiple con evaluación compleja

- *Patrón de unión sincronizada:* marca un punto donde múltiples rutas convergen en una única salida bajo la condición de sincronización, esto significa que el proceso espera a que se completen todas las rutas. Se usa un punto de decisión incluido en modo unión.

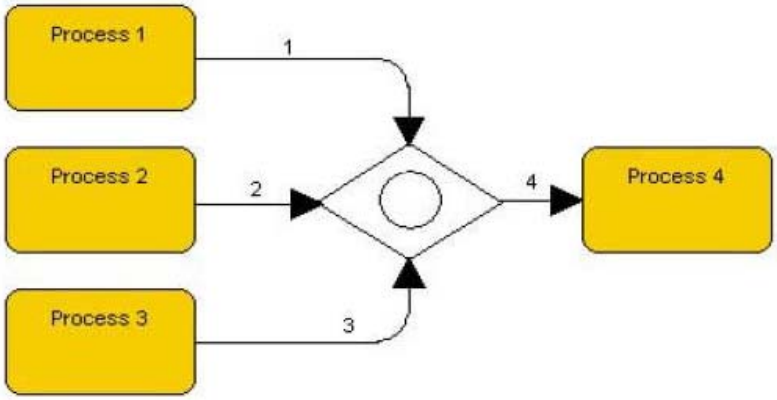


Ilustración 39: Patrón de unión sincronizada en BPMN

Cuando la unión evalúa una expresión para dejar continuar se utiliza puntos de decisión complejos.

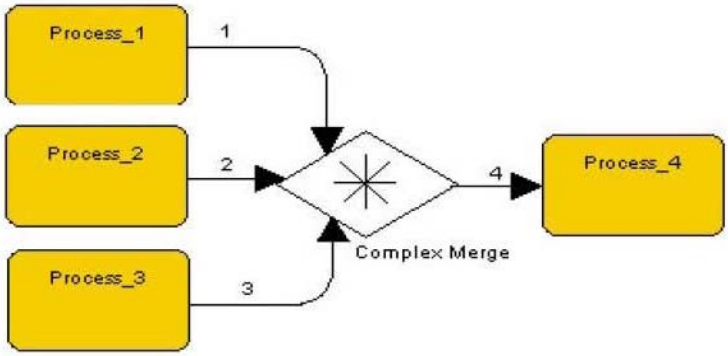


Ilustración 40: Patrón de unión sincronizada con evaluación compleja

- *Patrón de ciclos arbitrarios:* consiste en puntos de decisión que hacen que se vuelva a una actividad anterior del proceso para que parte de él se repita.

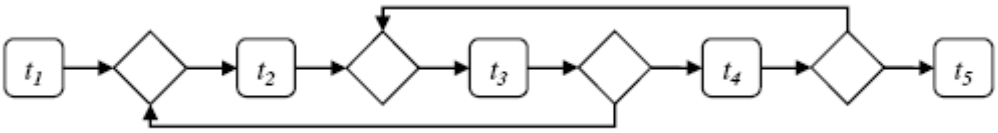


Ilustración 41: Patrón de ciclos arbitrarios en BPMN

El lenguaje Maude

Maude es un lenguaje de programación que implementa las reglas que expresan la transformación de unos estados que representan a los modelos de computación. La unidad básica para la especificación de estos modelos se realiza a través de *módulos*, donde se declara la sintaxis que establece el lenguaje que describe al sistema.

A continuación se van a introducir los conceptos y la sintaxis de los módulos necesarios para poder definir especificaciones de tipos de datos sencillos. El lenguaje proporciona muchas más posibilidades de las que aquí se van a tratar.

Módulos funcionales

Los módulos funcionales de Maude especifican los tipos de datos y operaciones que definen al sistema. Un ejemplo de módulo funcional que define a los números naturales junto a la operación factorial:

```
fmod NAT+FACT is
   protecting BOOL .

   sort Nat .

   *** constructores
   op 0 : -> Nat [ctor] .
   op s : Nat -> Nat [ctor] .                    *** Operador prefijo

   vars N M : Nat .

   *** suma
   op _+_ : Nat Nat -> Nat [comm assoc id: 0] .*** Operador infijo
   eq N + 0 = N .
   eq N + s(M) = s(N + M) .

   *** factorial
   op _! : Nat -> Nat .
   ceq (s N) ! = (s N) * N ! if 0 < N .  *** Ecuación condicional
   eq N ! = 1 [owise] .
endfm
```

Los módulos funcionales empiezan con la palabra reservada *fmod*, seguido del nombre del módulo y la palabra reservada *is*.

A continuación se importan los módulos necesarios utilizando las palabras reservadas *protecting*, *extending* o *including,* seguidas del nombre del módulo a importar. Informalmente, *protecting* exige que los tipos presentes en el módulo importado no se alteren con la inclusión de nuevos elementos ni con la identificación de elementos que anteriormente eran distintos; *extending* permite la creación de nuevos elementos en un tipo, pero sigue sin permitir la identificación de elementos distintos ya existentes; *including* no impone ningún tipo de restricciones. El sistema, sin embargo, no comprueba si estas restricciones realmente se satisfacen, por lo que desde un punto de vista operacional no existe ninguna diferencia entre ellas.

Tras las importaciones y precedido de la palabra *sort*, aparece el nombre del tipo de datos que se va a definir.

Los operadores se declaran utilizando la palabra *op,* seguida del nombre de la operación y los tipos de sus argumentos, después se escribe una flecha (->) y el tipo del resultado de la operación. Maude permite el uso de una sintaxis prefija o infija. En este último caso, los símbolos de subrayado indican la posición de los argumentos que deben estar en correspondencia con la aridad del operador. Además los operadores se pueden declarar con los siguientes atributos: conmutativo (*comm*), asociativo (*assoc*), elemento identidad (*id*) o constructor (*ctor*).

Si en la definición de las ecuaciones hacen falta variables, estas aparecen precedidas de la palabra *var* (o *vars* si se definen varias variables del mismo tipo). Tras el nombre se ponen dos puntos (:) y el tipo de la variable.

Por último se escriben las ecuaciones, que van precedidas de la palabra *eq*. Si son condicionales van precedidas de la palabra *ceq* y al final llevan la palabra *if* seguida de la condición. Las condiciones se escriben comprobando si los términos son iguales (= =) o diferentes (=/=), o uniendo expresiones booleanas con la conectiva conjuntiva (/\). Existe la opción de añadir la palabra *owise* al final de una ecuación sin condición, indicando que se ejecutará sólo cuando se dé un caso que no esté cubierto por ninguna de las restantes ecuaciones condicionales.

El módulo termina con la palabra *endfm*.

Módulos de sistema

Los módulos de sistema son una extensión de los módulos funcionales que permiten la declaración de reglas de reescritura, que especifican una transición que tiene lugar en el sistema. Es decir, si el patrón del lado izquierdo de la regla encaja con un fragmento del estado del sistema y la condición de la regla, si la hay, se satisface, entonces la transición toma lugar y el fragmento encajado es transformado en la correspondiente instancia del lado derecho de la regla.

Su sintaxis es la misma que la de los módulos funcionales, excepto por su cabecera y cierre, que utiliza las palabras reservadas *mod* y *endm*, y a las que se añade *rl* y *crl* para introducir reglas de reescritura.

```
mod FACT is
      protecting NAT+FACT .

      var N : Nat .
      rl [fact] N => N ! .
endm
```

Para reescribir un término en un módulo de sistema utilizando las reglas junto con las ecuaciones se utiliza la orden *rewrite* junto con el término que representa al sistema que se quiere estudiar.

```
Maude> rewrite s(s(s(0))) .
result Nat: s(s(s(s(s(0))))
```

Módulos orientados a objetos

Un sistema orientado a objetos es un multiconjunto de objetos y mensajes, con las propiedades asociativa, conmutativa e identidad (*none)* como atributos de los operadores, donde se usan reglas que describen los efectos de los eventos de comunicación entre los objetos y mensajes.

Para definir este tipo de sistema se requiere una extensión de Maude, llamada Full Maude, que incluye una especificación ejecutable que da soporte, entre otros, a los módulos orientados a objetos. En concreto, lo que hace es incluir al módulo

CONFIGURATION que define los tipos **Oid** para los identificadores de objetos, **Cid** para los identificadores de clases, **Object** para objetos y **Msg** para mensajes.

Este es un fragmento de código extraído del ejemplo del portal de revistas complutenses, en el anexo se puede ver el código completo:

```
(omod PREU is
     class Articulo | revista : String,
                      fasciculo : String,
                      codigo : String,
                      paginaI : String,
                      paginaF : String,
                      titulo : String,
                      descriptores : String,
                      resumen : String,

     var Art : Oid .
     vars Revista Fasc Codigo : String .
     vars PaginaI PaginaF Titulo Descriptores Resumen : String .

     rl [catalogarArticulo] :
          catArt(Revista, Fasc, Codigo, PaginaI, PaginaF,
                 Titulo, Descriptores, Resumen)
          < Art : Articulo | revista : Revista,
                             fasciculo : Fasc,
                             codigo : Codigo,
                             paginaI : PaginaI >
          =>
          < Art : Articulo | paginaF : PaginaF,
                             titulo : Titulo,
                             descriptores : Descriptores,
                             resumen : Resumen > .

endom)
```

Como se puede ver en el ejemplo la definición de un módulo orientado a objetos va contenida entre las marcas *omod* … *endom*, encerradas entre paréntesis para indicar que deben ser tratadas por Full Maude.

Para la declaración de una clase se utiliza la sintaxis *class* C **|** a1 **:** t1**, … ,** an **:** tn**,** donde *C* es el identificador de la clase, *ai* los atributos y *ti* los tipos de los valores de los atributos. Hay soporte de herencia entre clases con la instrucción *subclass* sC < C, donde *sC* es el identificador de la subclase y *C* el identificador de la clase.

Los objetos se representan como términos < O **:** C | a1 **:** v1**, … ,** an **:** vn >**,** donde *O* es el identificador del objeto, *C* es el identificador de la clase, *ai* son los identificadores de los atributos y *vi* los valores del atributo correspondiente.

La declaración de un mensaje se hace con *msg* m : Oid t1 … tn -> *Msg***,** donde *m* es el nombre del mensaje, el primer argumento *Oid* es el identificador del objeto destinatario y *ti* son los tipos del resto de argumentos.

El comportamiento asociado con los mensajes se especifica con la regla de reescritura *rl*: m(v1, … , vn) OI => OF**,** donde *m* es el nombre del mensaje, *vi* son los argumentos del mensaje, *OI* es el objeto destinatario antes de recibir el mensaje y *OF* es el objeto destinatario después de recibir el mensaje. Es posible no mencionar en una regla los atributos del objeto que no son relevantes para esa regla, de tal forma que:

- Los atributos mencionados solo en el lado izquierdo de la regla serán preservados sin cambios.
- Los valores originales de los atributos mencionados solo en el lado derecho de la regla son obviados.
- Todos los atributos no mencionados explícitamente no cambian.

Módulos de estrategia

El lenguaje de estrategias propuesto permite la definición de expresiones que controlan la forma en la que un término es reescrito, en un intento de controlar el no determinismo del proceso de ejecución.

El diseño está basado en una estricta separación entre las reglas de reescritura definidas en los módulos de sistema y las expresiones de estrategias establecidas en los módulos de estrategia. De hecho, esta separación hace posible definir diferentes módulos de estrategia que controlen de diferentes formas las reescrituras para un único módulo de sistema.

Una *estrategia* es descrita como una operación que, cuando es aplicada a un término, produce un conjunto de términos como resultado. El lenguaje de estrategias se compone de las siguientes expresiones:

Idle y fail

La estrategia i*dle* siempre tiene éxito sin modificar el término al cual es aplicado, y *fail* siempre falla dando un conjunto de resultados vacío.

Estrategias básicas

La estrategia básica de la forma *L[S]* consiste en la aplicación, en cualquier sitio del término donde encaje, de una regla del sistema identificada por su etiqueta *L*. Opcionalmente se pueden incluir variables en la regla para ser instanciadas por la sustitución *S,* para que el usuario tenga más control de la forma en que la regla es aplicada.

Top

*Top(*BE*)* restringe la aplicación de la estrategia básica BE a todo el término.

Test

xmatch T *s.t.* C es un chequeo de una propiedad que, cuando se aplica a un termino T', se satisface si hay un subtérmino de T' que encaja con el patrón T, y la condición C se cumple, aunque esta condición es opcional.

amatch T *s.t.* C obliga a que el patrón T encaje con todo el término.

Expresiones regulares

Corresponde a las construcciones de expresiones regulares de concatenación, unión e iteración. Los símbolos que los representan son los siguientes:

E ; E concatenación
E | E unión
E * iteraciones de 0 o más veces
E + iteraciones de 1 o más veces

Por ejemplo, la estrategia E; P (con P como test) filtra todos los resultados de E que no satisfacen la propiedad P.

Estrategias condicionales

El comportamiento de la estrategia *If* E *then* E' *else* E'' *fi* es que dado un término, la estrategia E es evaluada, si E se cumple, la estrategia E' es evaluada sobre los estados resultantes, en otro caso E'' es evaluada sobre el estado inicial. Por tanto, el primer argumento puede ser cualquier estrategia y en particular un test, que es un tipo de estrategia, por lo que se tiene el caso *If* P *then* E' *else* E'' *fi*, con P como test, donde la evaluación coincide con la típica distinción booleana.

Se pueden definir algunas combinaciones de estrategias como operaciones derivadas:

E *orelse* E' = *If* E *then idle else* E' *fi* evalúa E en un estado, si la evaluación se satisface, el resultado es el obtenido, pero si falla, entonces E' es evaluada en el estado inicial.

Not(E) = *If E then fail else idle fi* invierte el resultado de evaluar E, por tanto *not*(E) falla cuando E es evaluada con éxito y viceversa.

E! = E *; *not*(E) y E! = *If E then* E! *else idle fi* ambas ecuaciones definen la estrategia de repetir E hasta que ya no se satisfaga.

Try(E) = *If* E *then idle else idle fi* evalúa E en un estado; si se satisface, el correspondiente resultado es dado, pero si falla, el estado inicial es devuelto.

Test(E) = *If not*(E) *then fail else idle fi* y *test*(E) = *not*(*not*(E)) ambas ecuaciones son equivalentes y chequean si se satisface o no el resultado de E, pero no cambian el estado inicial.

Reescritura de subtérminos

Se puede tener más control sobre la forma en la que los subtérminos son reescritos con la estrategia *amatchrew* T *s.t.* C *by* T1 *using* E1,…, Tn *using* En que para aplicarse a un término T', primero se selecciona un subtérmino de T' que encaje con T y cumpla C, aunque esta condición puede ser omitida si se quiere. Entonces, los términos T1,…, Tn,

los cuales deben ser disjuntos en T, son instanciados apropiadamente y reescritos tal y como describen las estrategias E1,…, En respectivamente. Los resultados son combinados en T y sustituidos en T'.

La versión *matrew* trabaja de la misma forma, pero se ejecuta solo cuando encaja con todo el término.

Recursión

La recursión se consigue dando un nombre a una estrategia y usando este nombre en la propia estrategia.

Módulos de estrategia y comandos

Los módulos de estrategias tienen la siguiente forma:

```
smod STRAT is
     protecting M .
     strat E1 : T11 ... T1m @ K1 .
     sd E1(P11,...,P1m) := Exp1 .
          ...
     strat En : Tn1 ... Tnp @ Kn .
     sd En(Pn1,...,Pnp) := Expn .
     csd En(Qn1,...,Qnp) := Expn' if C .
endsd
```

donde *M* es el módulo del sistema cuyas reescrituras serán controladas,

E1,…,En son identificadores de estrategias,

Exp1,… Expn son las estrategias sobre las reglas del módulo M.

Un identificador de estrategia puede tener argumentos, que son usados para capturar los valores pasados cuando la estrategia es invocada. Para ello la estrategia es declarada con la palabra clave *strat*, y el tipo de los argumentos es especificado entre los símbolos : y @. Después del símbolo @, el tipo del término al cual esta estrategia puede ser aplicada es también especificado.

Una definición de estrategia, introducida con la palabra clave *sd*, asocia una estrategia con un identificador de estrategia. Estas definiciones pueden ser condicionales, usando la palabra clave *csd,* así puede haber varias definiciones para el mismo identificador de

estrategia, aunque deben referirse a casos disjuntos, ya sea porque la condición usada difiere o porque cambien los argumentos.

La idea básica es que estas declaraciones de estrategias suministren abreviaciones para que el usuario pueda utilizar las estrategias en un comando de reescritura de la forma *srew* T *using* E, donde se reescribe un término T usando una estrategia E.

Ejemplo: Portal de revistas científicas complutenses

Definición del sistema

Los diferentes departamentos de la universidad editan publicaciones en papel con trabajos, investigaciones, etc. que realizan sobre la materia que en cada caso les compete. El Servicio de Publicaciones en colaboración con la Biblioteca ha decidido comenzar con la digitalización de dicho material con dos ideas fundamentales: la preservación y difusión de dicho material a la comunidad universitaria.

En la actualidad se cuenta con más de 60 revistas digitalizadas, realizándose archivos independientes a nivel de artículo. Se han digitalizado todos los números disponibles de cada una de las revistas y el proceso de digitalización de los nuevos números seguirá de forma continuada en el tiempo. El formato elegido para publicar esta información es el formato .pdf

Así pues, el objetivo principal del proyecto consiste en desarrollar una aplicación que permita:

- la *catalogación* (preservación) de los ficheros pdf obtenidos de la digitalización de artículos de revista,
- la *consulta* (difusión) de los contenidos generados a través de la catalogación anterior.

Diagrama de contexto

En el diagrama de contexto se muestran los usuarios que participan tanto en la elaboración del contenido como en la explotación del mismo.

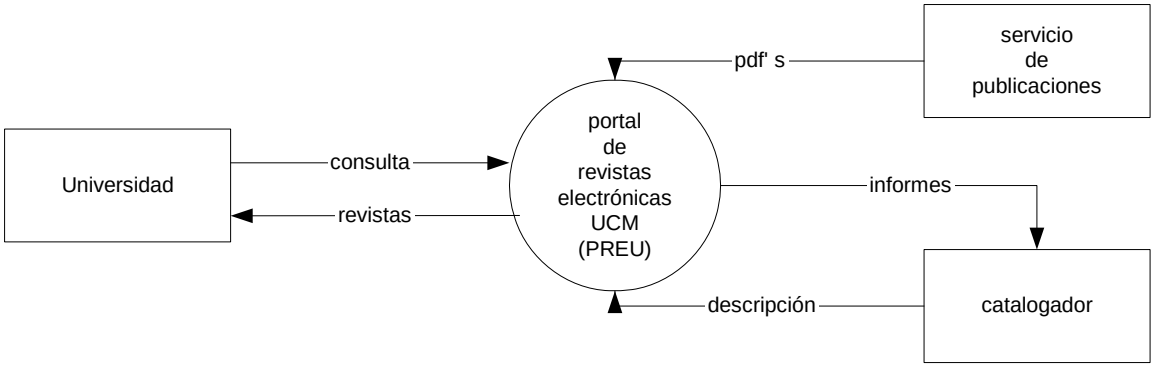


Ilustración 42: Diagrama de contexto

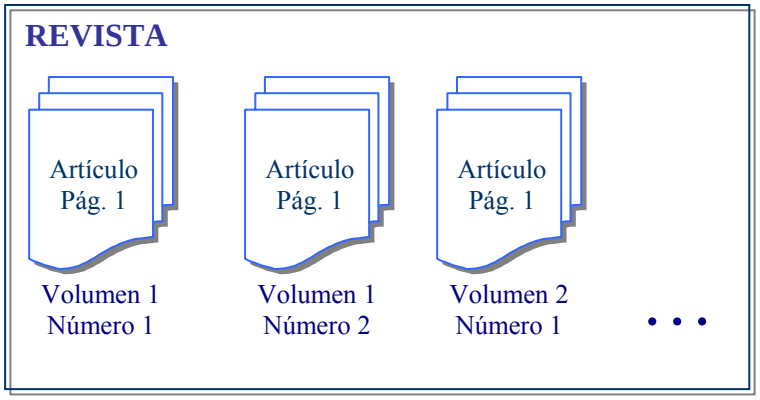
Se identifican las siguientes entidades externas:

- *Servicio de publicaciones*: es el proveedor de las revistas en formato pdf.
- *Catalogador*: personal de la biblioteca que realiza la catalogación de los artículos.
- La *comunidad universitaria* como usuario final que consulta la información.

Estructura del sistema

Los tres elementos principales que forman la estructura jerárquica del sistema son:

- *Revista*: publicación por fascículos editada con periodicidad que trata sobre una materia.
- *Fascículo*: cada uno de los cuadernos impresos que publica la revista. Quedan organizados por el volumen y/o número asignado de manera correlativa. A veces se publican fascículos con ocasión de un motivo especial y fuera de la periodicidad normal de la revista a los que se les llama *Anejos*.
- *Artículo*: cada uno de los escritos que se publican en los fascículos con los trabajos de los departamentos. Se identifica por la página del fascículo en la que ha sido publicado.

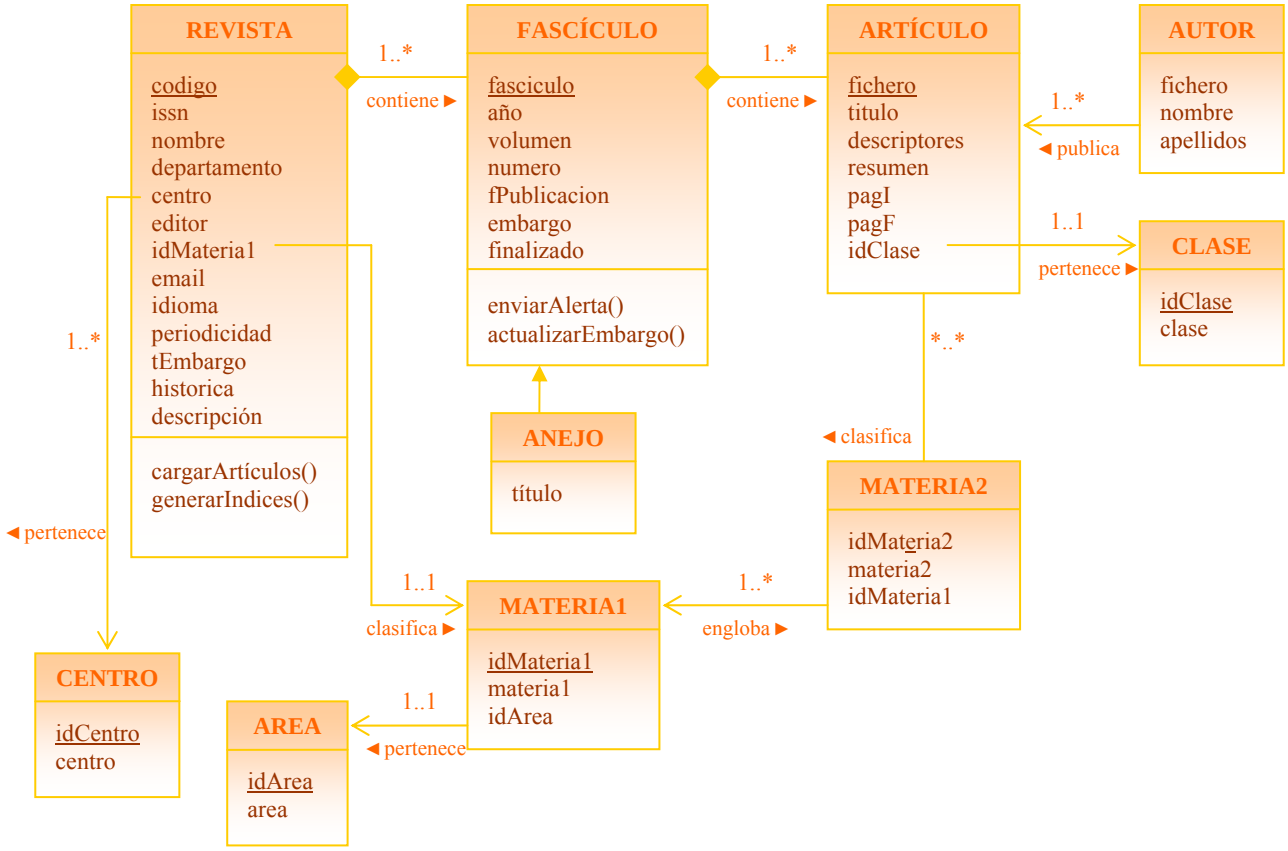


Estructura jerárquica del sistema

Establecimiento de requisitos

Requisitos funcionales que afectan al modelo de base de datos

- Un artículo pertenecerá exclusivamente a una de las clases definidos.
- Un artículo puede estar escrito por uno o más autores.
- Un autor puede publicar uno o más artículos.
- El nombre del fichero identifica de forma unívoca un artículo (ver ejemplo de codificación más adelante).
- Un artículo sólo pertenece a un fascículo.
- Un fascículo sólo pertenece a una revista.
- El anejo es un tipo de fascículo.
- Se utilizará como identificador de fascículo al que pertenece el artículo los 16 primeros caracteres del nombre del fichero.
- Los cuatro primeros caracteres del nombre del fichero se utilizarán como identificadores de la revista a la que pertenece el artículo.
- Las revistas proceden de un centro.
- Una revista pertenece a una materia principal.
- Las materias principales están agrupadas en 4 áreas: humanidades, ciencias, ciencias de la salud y ciencias sociales.
- Un artículo se describe por una o más materias secundarias.



Elaboración del modelo de datos: Diagrama de clases UML

Formato del nombre del fichero para los artículos

A continuación se muestra un ejemplo de cómo codificar el nombre del fichero PDF de un artículo:

REFA19930001 0003 0120

FASCÍCULO

REFA	Posición **1 - 4**	Código de la Revista
1993	Posición **5 - 4**	Año del Volumen
0001	Posición **9 - 4**	Volumen
0003	Posición **13 - 4**	Número

0120	Posición **17 - 4**	Número de página donde comienza el artículo

Requisitos de seguridad

El acceso es abierto, sin embargo algunas revistas tendrán un periodo de embargo en los últimos números. El fascículo no se verá hasta que dicha fecha se haya cumplido.

Funcionalidades

La aplicación contará con tres módulos:

Módulo de catalogación

Se encarga de recolectar y almacenar los metadatos referentes a las revistas y a los artículos disponibles. Desde la catalogación se podrán realizar las operaciones de:

- Añadir una nueva revista
- Modificación de los datos de las revistas
- Modificación de los datos de los artículos
- Modificación de los datos de los fascículos
- Dar de alta nuevos artículos en una revista
- Envío de alertas donde se avisa al usuario de las nuevas publicaciones

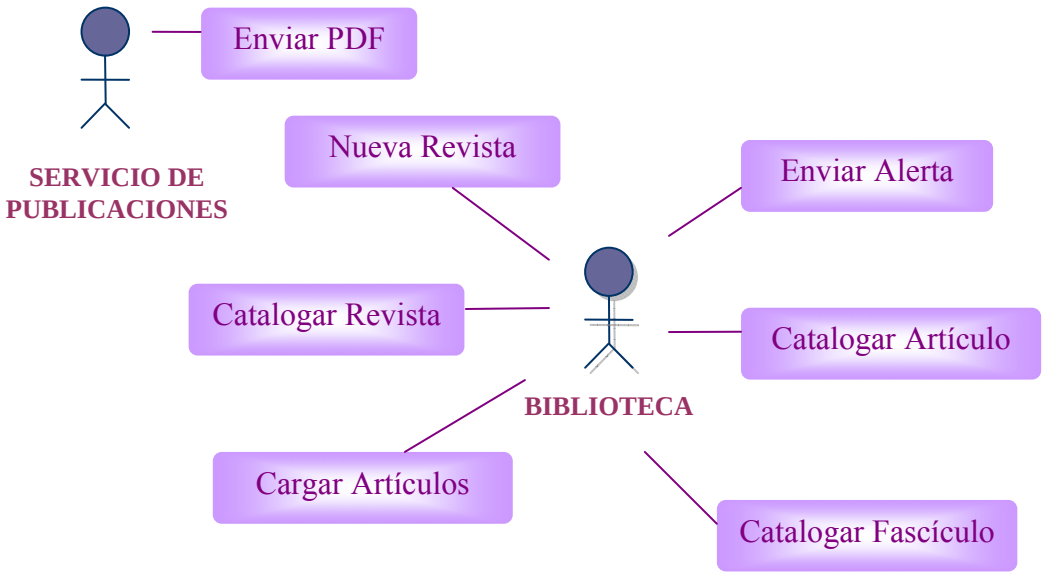
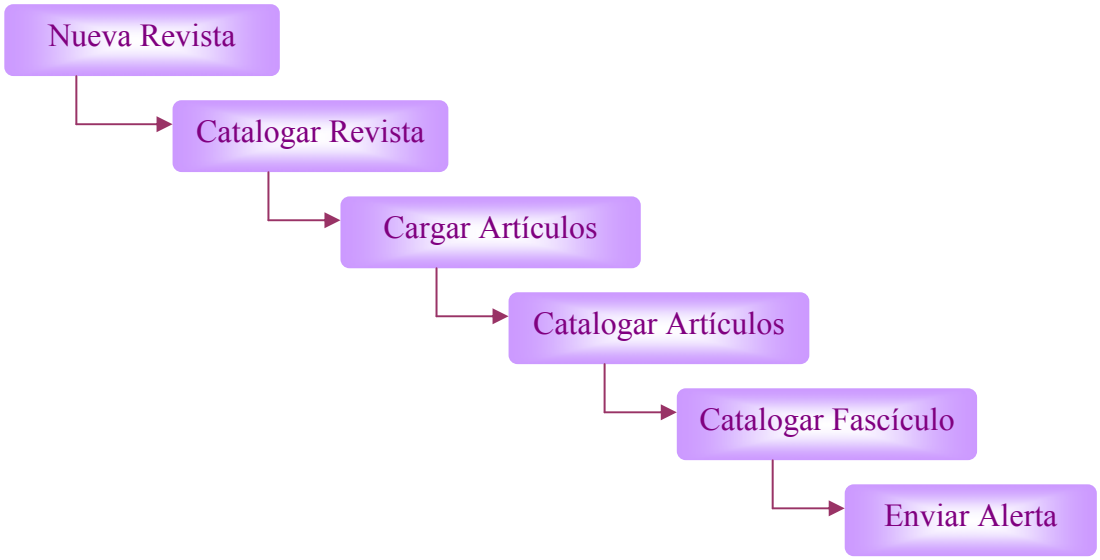


Diagrama de casos de uso en UML para el módulo de catalogación

El orden en el que se realizan las operaciones de catalogación es importante ya que para añadir nuevos artículos a una revista, ésta debe estar dada de alta con anterioridad y preferiblemente con toda su información completa, pues de lo contrario no se podrá tener acceso a ella a través del portal. Durante el proceso de carga de los artículos también se realiza la incorporación del fascículo al que pertenece, en caso de que aun no esté dado de alta. Es por eso que los pasos siguientes son los de catalogar a los artículos cargados y al fascículo generado si éste fue creado durante la carga. Solo después de todo este proceso se puede considerar al fascículo por finalizado, permitiendo el envío de las alertas con las novedades a los usuarios suscritos.



Orden de ejecución de las operaciones del módulo de catalogación

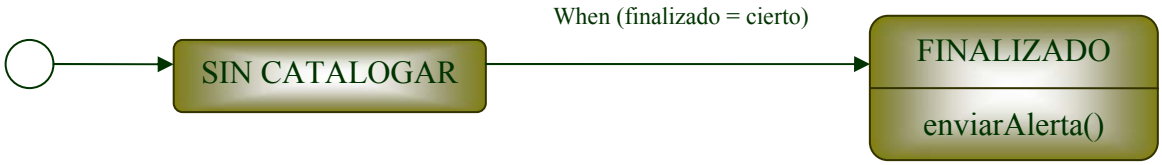


Diagrama de estados UML: Finalizar fascículo

Módulo de consulta

Permite consultar las revistas y artículos catalogados. Se pueden diferenciar dos submódulos:

Portal de revistas

Permite el acceso a todas las revistas. Se podrán realizar búsquedas por revistas sobre los metadatos. También se podrán listar las revistas alfabéticamente, por área o por materia.

Acceso a la revista

Al entrar en la revista se podrá acceder a todos sus fascículos, así como realizar búsquedas en los artículos de dicha revista. Dispondrá de un índice alfabético por autor y otro por título de todos sus artículos. Los usuarios además podrán suscribirse a la revista para mantenerse informado de las novedades en sus publicaciones.

Diagrama de casos de uso en UML para el módulo de consultas

El modo habitual para acceder a un artículo comienza por entrar en el portal de revistas y buscar la revista que se desea consultar a través de los listados o bien buscando por palabra clave. Una vez dentro de la revista se busca el artículo por palabra clave o a través de los índices de título y autor o accediendo a los fascículos que forman la revista.

Orden de búsqueda en el modulo de consultas

Módulo del sistema

El sistema ejecuta diariamente una tarea programada para actualizar el embargo de todos los fascículos. Se calcula sumando los meses de embargo que tiene asignada la revista a la fecha de publicación del fascículo y comprobando si se ha superado la fecha actual.

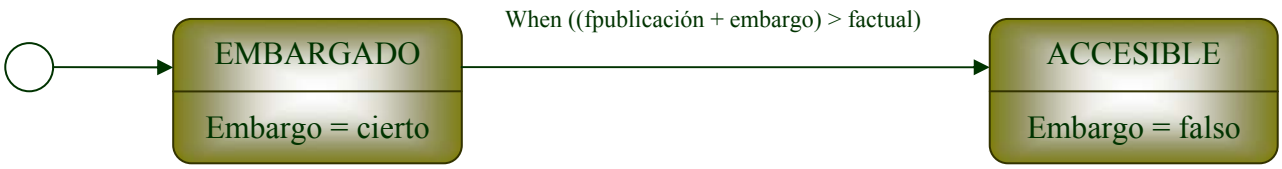


Diagrama de estados UML: Calcular embargo

Además semanalmente genera los índices con los títulos de los artículos y los autores para cada una de las revistas.

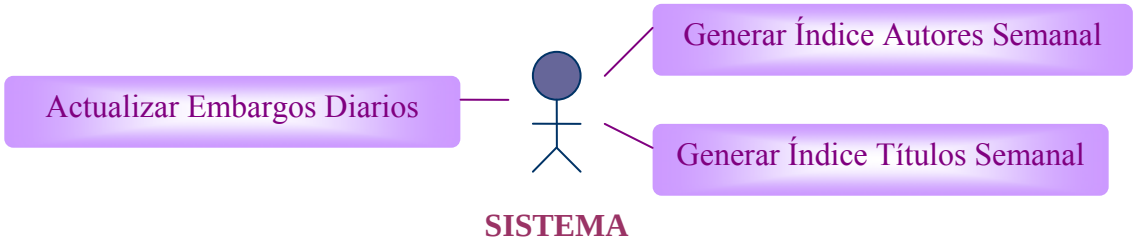


Diagrama de casos de uso en UML del módulo del sistema

Diseño del sistema: diagramas de actividades en UML y BPMN

A continuación se muestran los diagramas en UML y BPMN de tres procesos que se usaron como guía durante la implementación del proyecto.

Dar de alta una nueva revista

Para dar de alta una revista primero se han de introducir algunos datos representativos, como el código que la identifica, el ISSN asignado, su nombre y el centro que la publica. Todos los valores han de ser comprobados porque en el caso de que alguno no sea correcto se rechaza el proceso de alta, pero la comprobación más importante que debe hacerse es la de corroborar que el código asignado sea único, es decir, no esté concedido a otra revista. Solo si cumple con todos los requisitos se procede a guardar los datos en la base de datos. Este procedimiento se representa gráficamente a través de los siguientes diagramas:

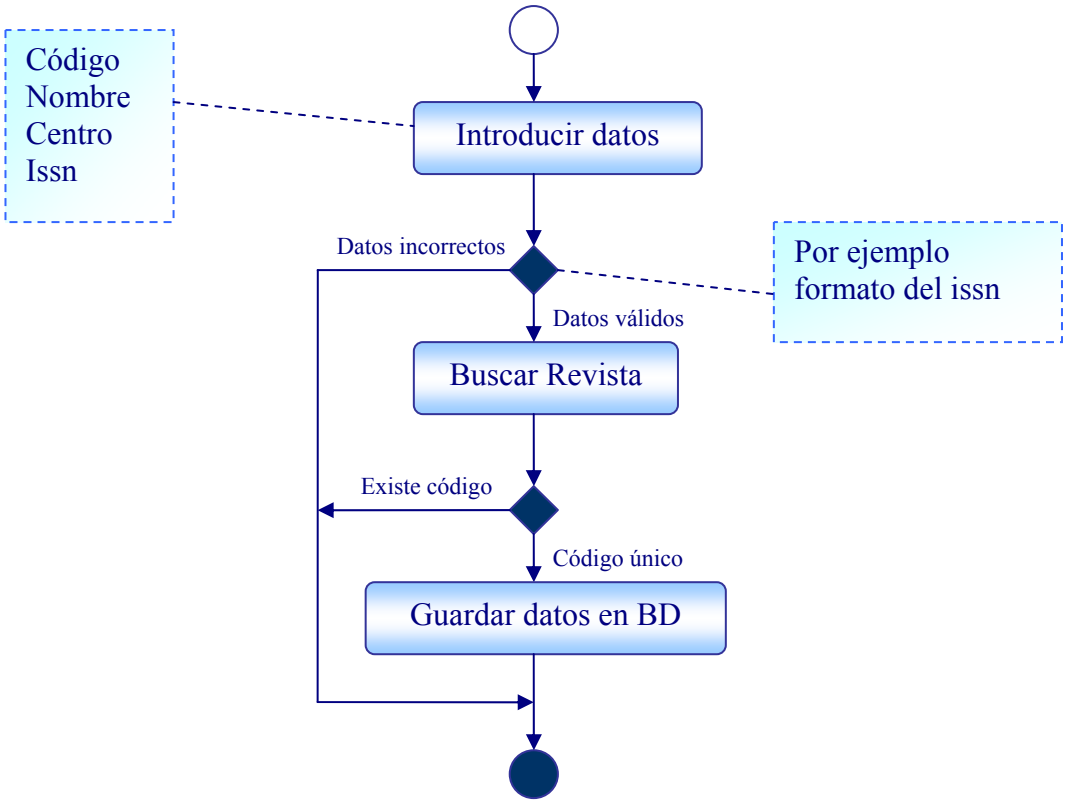


Diagrama de actividades en UML con el alta de la revista

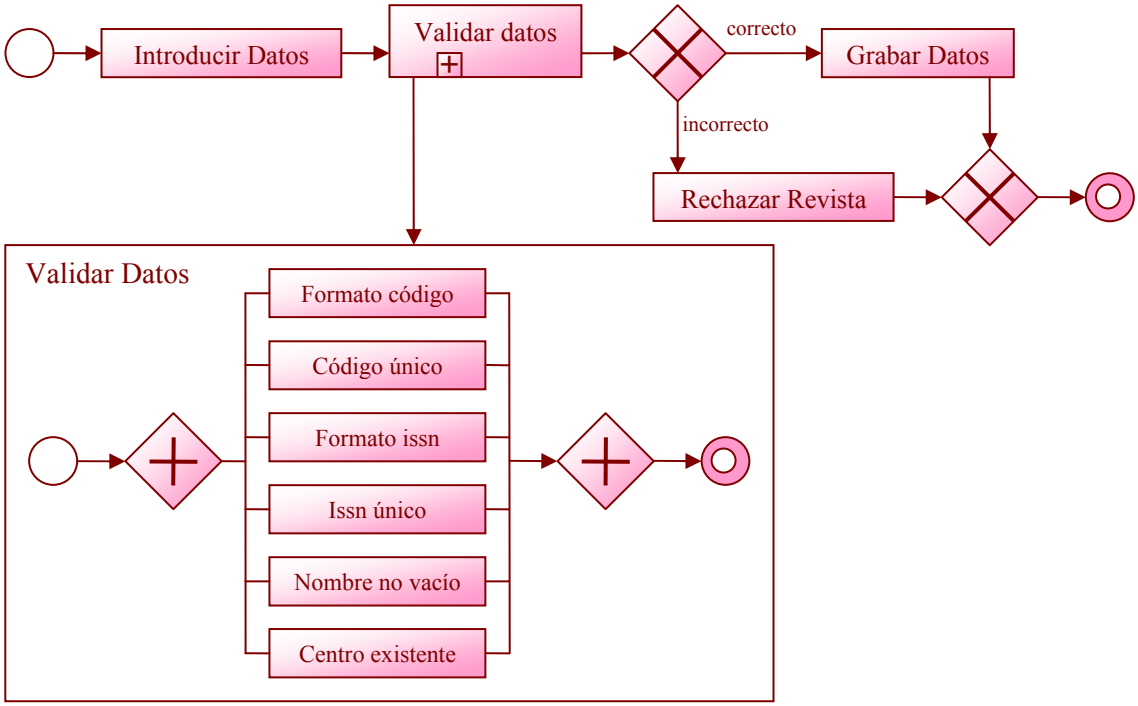


Diagrama en BPMN para el alta de la revista

Catalogar Fascículo

Un proceso algo más complicado es la catalogación de un fascículo que ya existe. Lo primero que se hace es buscarlo de entre los añadidos a la revista a la que pertenece. Una vez seleccionado se permite modificar cualquier dato que ya tuviera guardado, pero para la consistencia de la base de datos hay que tener en cuenta que si se produce algún cambio en el año, volumen o número, el nombre del fichero que identifica al fascículo y a sus correspondientes artículos cambia, pues están formados por la concatenación de éstos (ver formato del nombre de los artículos), por lo que deberán ser todos actualizados. Además se debe recalcular el embargo si la fecha de publicación es modificada. Cuando la catalogación del fascículo se dé por finalizada, lo que también incluye el que sus artículos hayan sido catalogados, hay que enviar el correo con toda la información a los suscriptores. Los diagramas que lo representan en formato gráfico:

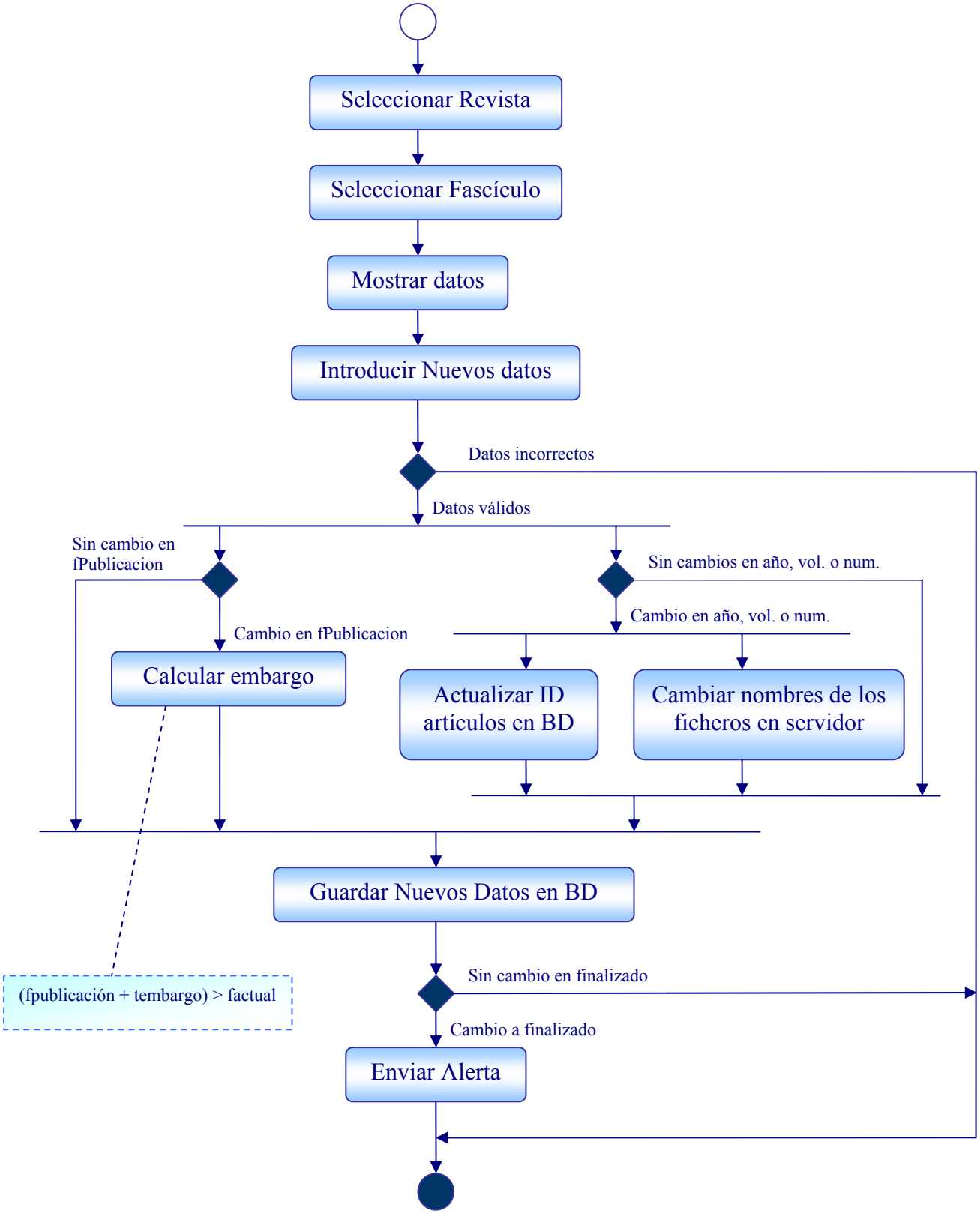


Diagrama de actividades en UML con la catalogación de un fascículo

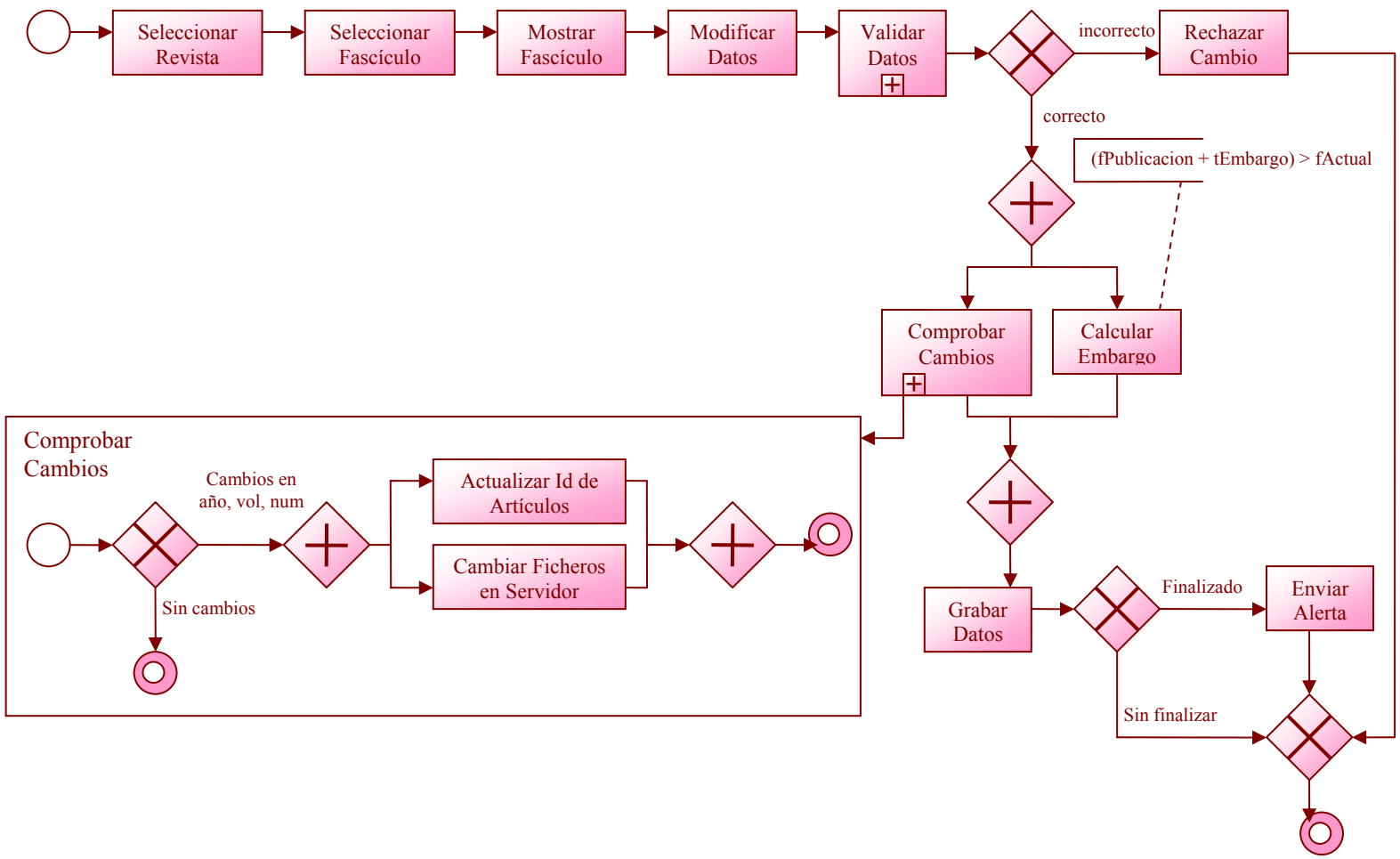


Diagrama en BPMN para la catalogación de un fascículo

Cargar artículos en una revista

Otro procedimiento interesante es el de añadir nuevos artículos. Lo primero que hay que hacer para realizar la carga en una revista ya seleccionada es indicar el fascículo al cual pertenecen los artículos. Si ya existiera, los ficheros con los artículos seleccionados irán asociados a ese fascículo; sino, un nuevo fascículo será dado de alta. En este último caso es obligatorio introducir la fecha de publicación para calcular el embargo, antes de que los artículos se guarden en la base de datos, evitando que se hagan públicos antes de tiempo. Inicialmente es suficiente con indicar las páginas de inicio y fin en cada uno de los artículos, ya que el resto de datos se rellenarán durante el proceso de catalogación. Si los valores son correctos entonces se guardan los datos de los artículos y del fascículo si no existía previamente.

A continuación se muestra gráficamente el proceso con diagramas. Merece la pena indicar que en el caso de BPMN se incluye el subproceso compuesto *Guardar Artículos*, que podría ejecutarse con múltiples instancias en paralelo, cada una con un artículo, y que solo alcanzaría la siguiente actividad si todos los artículos se guardan correctamente.

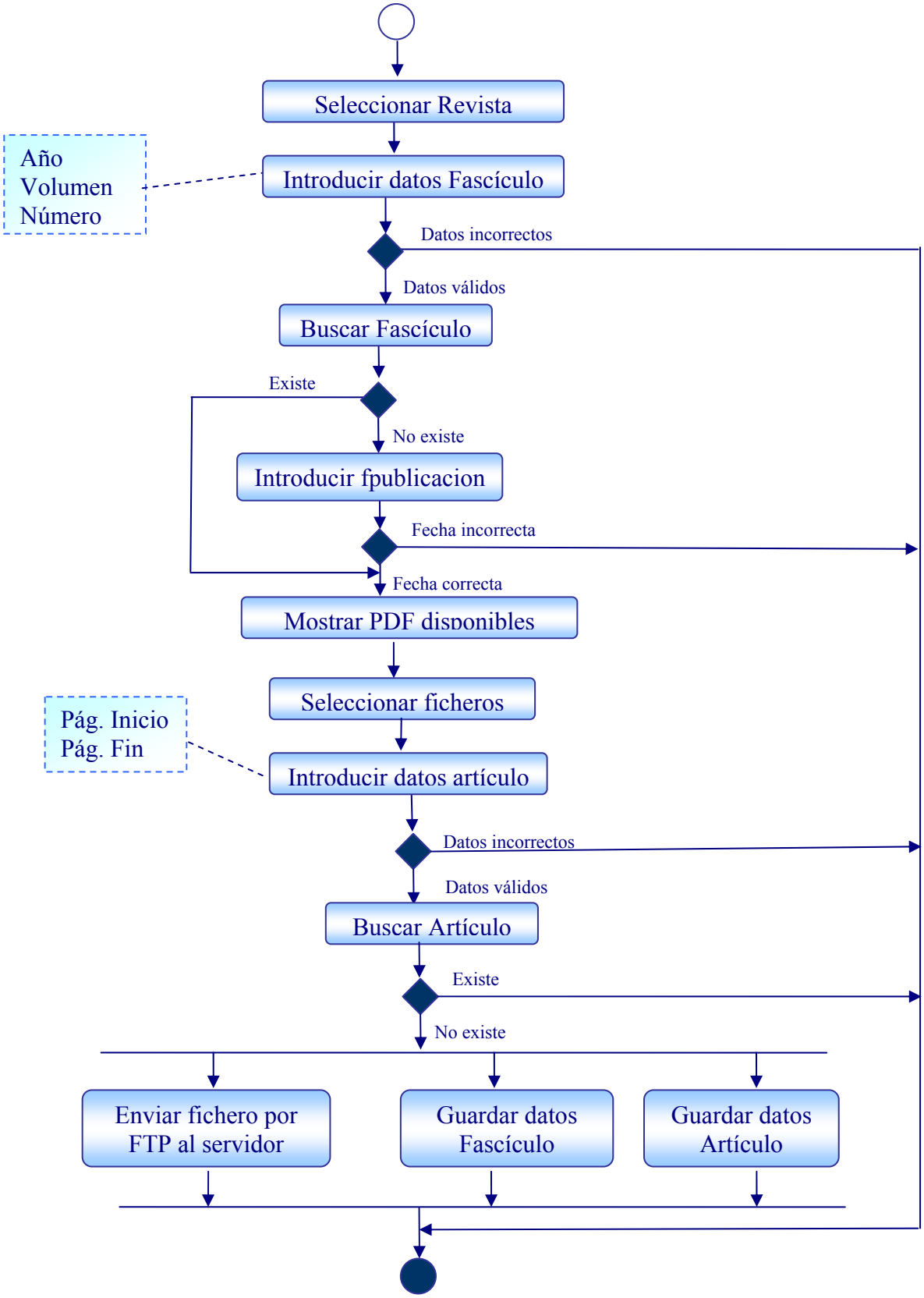


Diagrama de actividades en UML con la carga de artículos en una revista

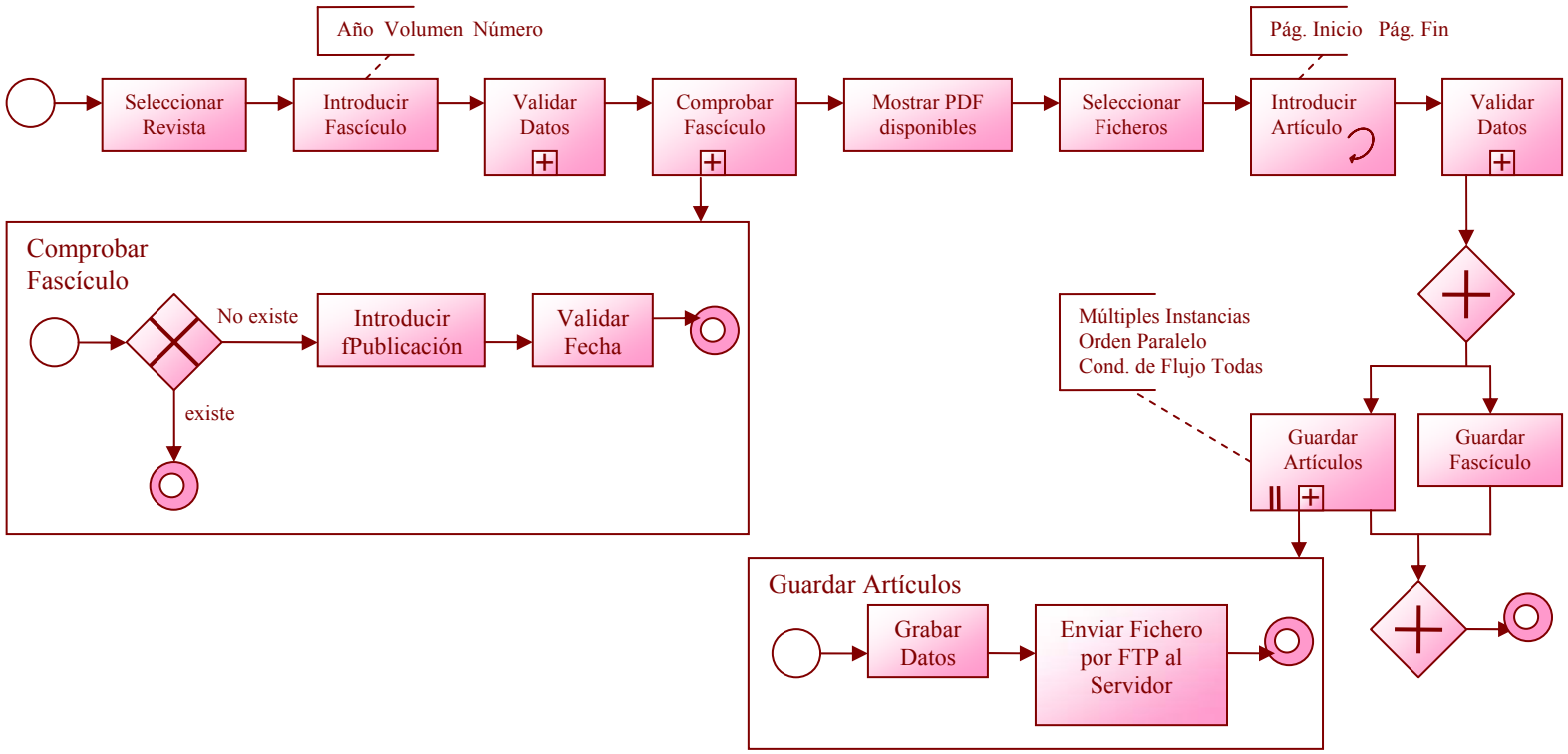


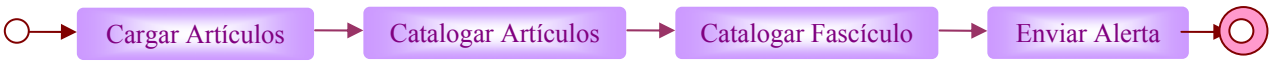
Diagrama en BPMN para la carga de artículos en una revista

BPMN & Estrategias Maude

La relación entre BPMN y las estrategias de Maude no vino impuesta sino que surgió de la necesidad de coordinar la ejecución de las reglas para la catalogación, ya que una vez estuvieron definidas y probadas individualmente, al ir a realizar las pruebas en conjunto para probar el proceso de catalogación al completo, en el resultado aparecían muchos mensajes de error, indicando que no era posible catalogar los artículos, debido a que no existían cuando se suponía que habían sido dados de alta. El motivo se debía a que las reglas se ejecutaban en otro orden al que nosotros necesitamos, intentando catalogar artículos antes de cargarlos, y la solución pasaba por incluir estrategias que guiaran al proceso. Pero esta guía tiene muchas formas de entenderse y a continuación se muestran algunas de las posibilidades que había a la hora de realizar la catalogación.

Antes de empezar a comentar las posibles combinaciones, debemos mencionar la correspondencia directa entre los elementos básicos de ambos lenguajes, es decir, una tarea BPMN que ejecuta una acción simple, se traduce en la estrategia básica de ejecutar una regla (*RL*) en Maude; mientras que el subproceso en BPMN formado por un conjunto de actividades, equivale en Maude a la composición de estrategias (E_1; E_2). Y ahora sí damos paso a explicar las propuestas de catalogación.

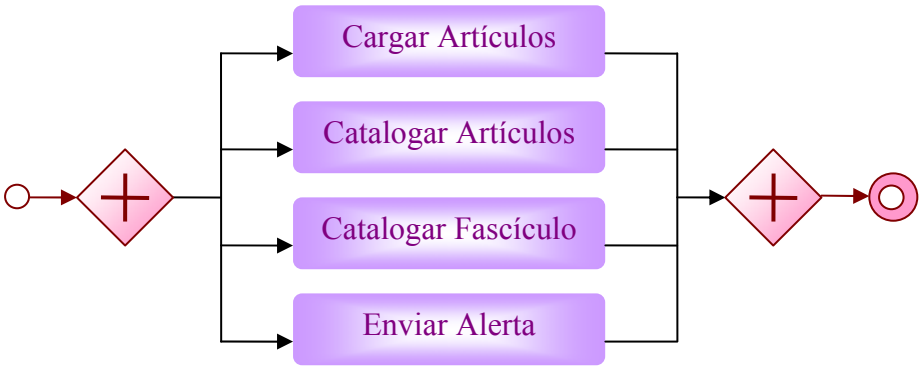
Una de las estrategias que se podría haber llevado a cabo en el proceso de catalogación, es seguir el orden de cargar los artículos pertenecientes a un fascículo, catalogarlos, catalogar al fascículo y enviar la alerta, es decir, completar el fascículo antes de pasar al siguiente. Para su representación BPMN usa el flujo secuencial que ordena la ejecución de las actividades y Maude la concatenación de las reglas de catalogación (RL_1; RL_2):




```
strat catalogar : String @ Configuration .
sd catalogar := cargarArticulos; catalogarArticulo;
                calogarFasciculo; enviarAlerta .
```

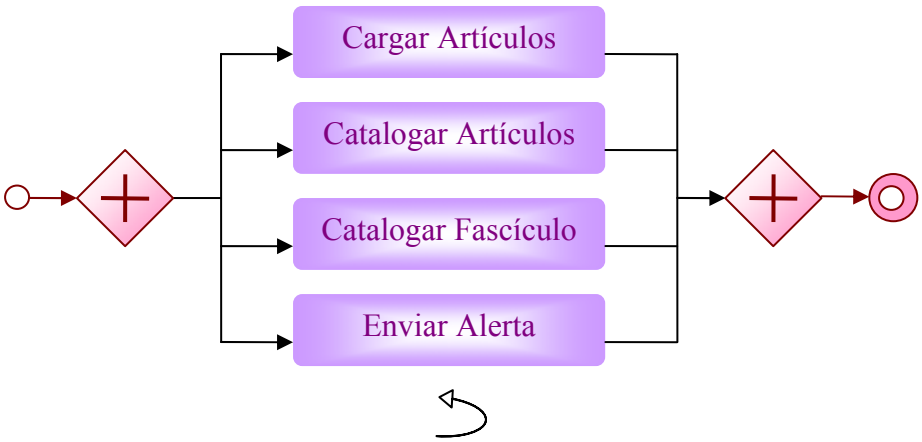
Otra alternativa, que ha sido la implementada en el ejemplo del proyecto, es realizar la catalogación independientemente del fascículo al que pertenezcan los artículos, es decir,

el proceso comienza por cargar todos los artículos, cuando ya no haya más artículos que cargar entonces se catalogan todos, luego se catalogan los fascículos y por ultimo se envían juntas las alertas de todos los fascículos. En este caso BPMN usa un punto de decisión exclusivo donde solo la primera ruta que pueda ejecutarse continúa y se corresponde con la estrategia condicional Maude RL_1 *orelse* RL_2*:*



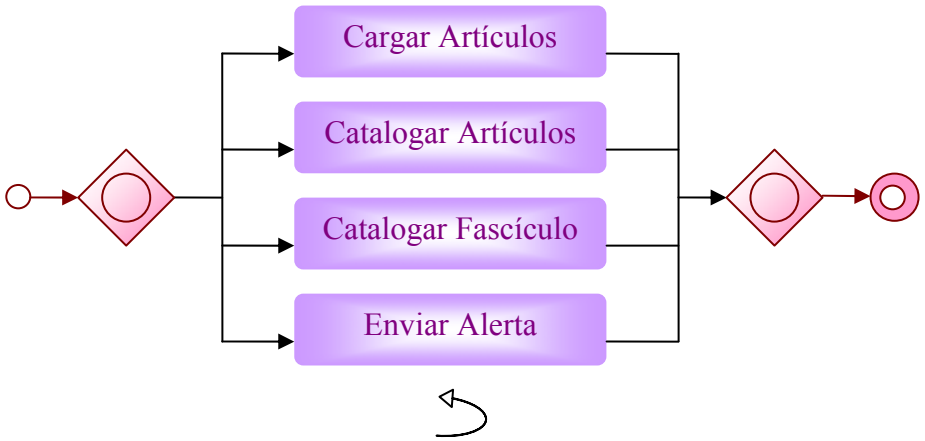

```
strat catalogar : @ Configuration .
sd catalogar := cargarArticulos
                orelse catalogarArticulo
                orelse catalogarFasciculo
                orelse enviarAlerta .
```

Ahora es necesario que la actividad y la estrategia se repitan, y vaya pasando por todas las pasos del procedimiento hasta que no haya más tareas que ejecutar. Para ello en BPMN hay que convertir la actividad en un subproceso de bucle, e implementar en Maude la repetición con alguna de las estrategias de iteración (*E** o *E+* o *E!*):



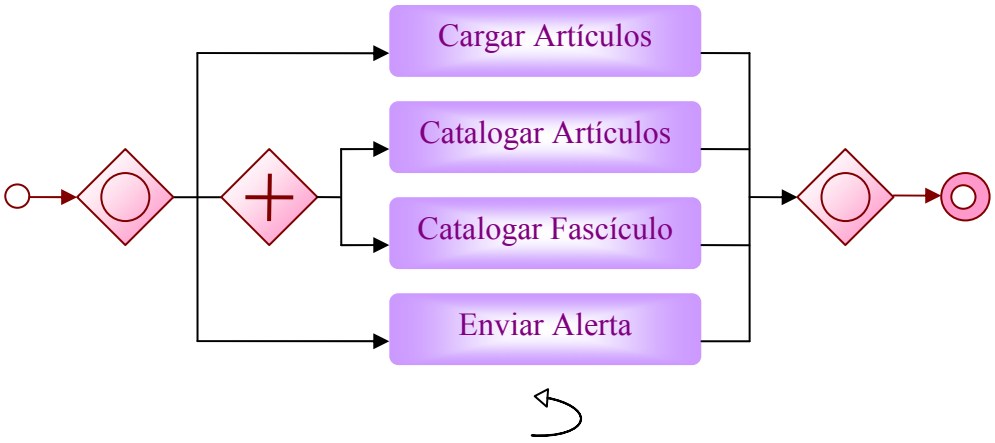
```
strat catalogar : @ Configuration .
sd catalogar := (cargarArticulos
                 orelse catalogarArticulo
                 orelse catalogarFasciculo
                 orelse enviarAlerta) ! .
```

Una última posible estrategia de catalogación es ejecutar a la vez todos los pasos que se puedan, es decir, que un fascículo puede darse de alta, mientras otro está catalogándose y otro enviando la alerta porque ya fue completado. Esto significa que cualquier ruta podría ejecutarse siempre que cumpliera las condiciones, lo que en BPMN se interpreta con un punto de decisión OR, y en Maude con la estrategia condicional *try(RL_1); try(RL_2):*



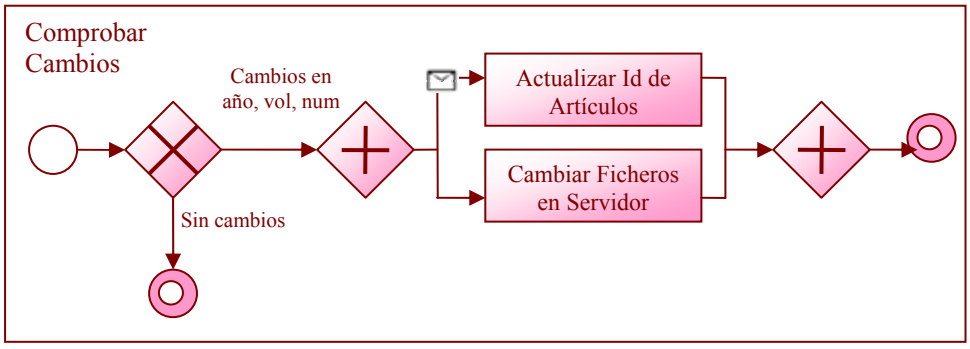

```
strat catalogar : @ Configuration .
sd catalogar := (try(cargarArticulos) ;
                 try(catalogarArticulo) ;
                 try(catalogarFasciculo) ;
                 try(enviarAlerta)) ! .
```

Como el orden de catalogación de los artículos y del fascículo no es importante, cualquiera de las dos actividades puede ejecutarse en el orden que se quiera. Esto se indica en BPMN con un punto de decisión paralelo o convirtiendo la actividad en un subproceso Ad hoc, mientras que en Maude se implementa con la estrategia de unión (E_1 | E_2):




```
strat catalogar : @ Configuration .
sd catalogar := (try(cargarArticulos) ;
                 try(catalogarArticulo | catalogarFasciculo) ;
                 try(enviarAlerta)) ! .
```

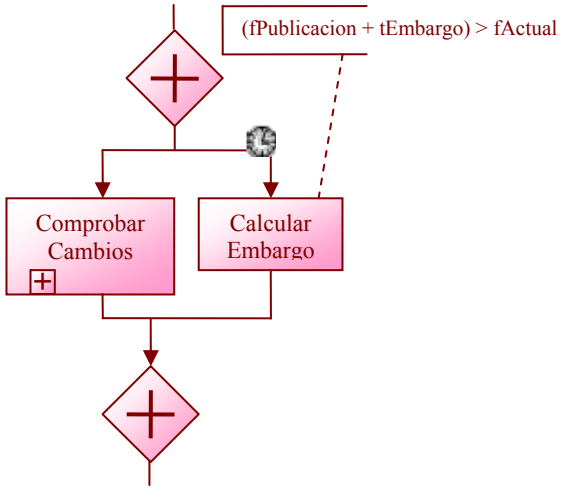
Sin embargo, los eventos en BPMN se asemejan más al envío de mensajes entre objetos en Maude. Por ejemplo, cuando en la catalogación de un fascículo ha habido cambios que afectan a su identificador, debe comunicárselo a sus correspondientes artículos para que actualicen los datos. Esta comunicación en BPMN se hace con un evento mensaje y en Maude poniendo un mensaje a los artículos:




```
crl [catalogarFasciculo] :
      catFas(Codigo, Fasc, Anno, Volumen, Numero)
      < Rev : Revista | codigo : Codigo,
                        listaFasciculos : (Fasc, ListaFasciculos) >
      < Fas : Fasciculo | codigo : Codigo,
                          fasciculo : Fasc,
                          anno : Ann,
                          volumen : Vol,
                          numero : Num,
                          listaArticulos : ListaArticulos >
      =>
```

```
      cambioFichero(Codigo + Anno + Volumen + Numero, ListaArticulos)
      < Rev : Revista | codigo : Codigo,
       listaFasciculos : (Codigo + Anno + Volumen + Numero, ListaFasciculos) >
      < Fas : Fasciculo | fasciculo : Codigo + Anno + Volumen + Numero,
                          anno : Anno,
                          volumen : Volumen,
                          numero : Numero >
      if (Anno =/= Ann or Volumen =/= Vol or Numero =/= Num)
        and length(Anno) == 4 and length(Volumen) == 4 and length(Numero) == 4
        and not (Codigo + Anno + Volumen + Numero) in ListaFasciculos .
```

Para los eventos de tiempo BPMN, la solución tomada en Maude ha sido introducir una variable con la fecha del día, que es enviada como argumento a través de los mensajes para las reglas que lo necesiten. Por ejemplo, cuando se introduce la fecha de publicación de un fascículo hay que calcular su embargo y para ello es necesario conocer la fecha actual; en BPMN se le envía a la actividad un evento con el tiempo y en Maude se le añade un valor más al mensaje:



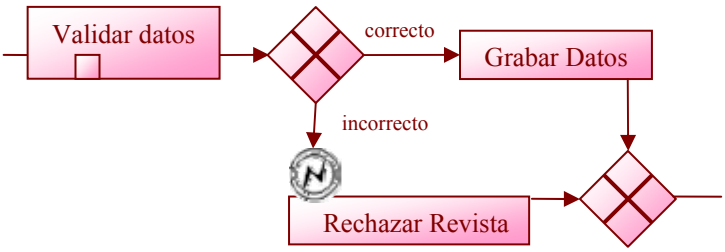

```
op hoy : -> Nat .
eq hoy = 20090316 .

catFas("REMA", "REMA200400180001", "2005", "0018", "0001", 200509276, hoy)

rl [catalogarFasciculo] :
      catFas(Codigo, Fasc, Anno, Volumen, Numero, FPublicacion, FActual)
      < Fas : Fasciculo | codigo : Codigo,
                          fasciculo : Fasc,
                          anno : Anno,
                          volumen : Volumen,
                          numero : Numero >
      =>
      embargo(Fasc, FActual)
      < Fas : Fasciculo | fpublicacion : FPublicacion > .
```

```
rl [actualizarEmbargos] :
      embargo(Fasc, FActual)
      < Fas : Fasciculo | codigo : Codigo,
                          fasciculo : Fasc,
                          fpublicacion : FPublicacion >
      < Rev : Revista | codigo : Codigo,
                        tembargo : TEmbargo >
      =>
      < Fas : Fasciculo | embargo : (FPublicacion + (TEmbargo * 100)) > FActual) >
      < Rev : Revista | > .
```

Otro tipo de evento que puede ser tratado en ambos lenguajes es la excepción. En Maude, por ejemplo, se pueden poner reglas que generen un mensaje de error que informe al usuario cada vez que cometa alguna irregularidad; como en el siguiente ejemplo, donde una vez introducidos los datos de una revista, se validan y si alguno es incorrecto, la revista es rechazada, lo que provoca un mensaje de excepción en BPMN, y un mensaje de error en Maude:

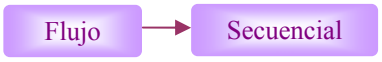
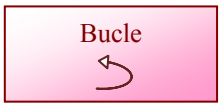



```
msg error : String -> Msg .

crl [nuevaRevista] :
      newRev(Codigo, ISSN, Nombre, Centro)
      < Por : Portal | listaRevistas : ListaRevistas,
                       listaCentros : ListaCentros >
      =>
      error("Error de validacion para la nueva revista " + Codigo )
      < Por : Portal | >
      if Codigo in ListaRevistas or length(Codigo) =/= 4
         or Nombre == "" or ListaCentros[Centro] == undefined .
```

En conclusión podemos sacar que, en términos generales, ambos se encargan de definir el control del comportamiento de los procesos, BPMN con una definición gráfica y Maude a través del lenguaje de estrategias que dirigen las reglas, es por ello que comparten muchos de los conceptos e ideas. Sin embargo, se ha de tener muy en cuenta que la interpretación de la notación gráfica de BPMN a código Maude tiene algunas limitaciones, dada la gran cantidad de sutilezas que aparecen en algunos de los símbolos. Este proyecto solo ha pretendido realizar un pequeño acercamiento entre ambas partes, quedando aun muchos elementos por relacionar.

Para finalizar resumimos las relaciones estudiadas entre los símbolos de BPMN y el lenguaje Maude en la siguiente tabla:

BPMN	Maude
	Regla *RL*
	Concatenación de estrategias $E_1; E_2$
	Concatenacion de reglas $RL_1; RL_2$
Punto de decisión exclusivo 	Estrategia condicional RL_1 *orelse* RL_2
Punto de decisión OR 	Estrategia condicional *try*(RL_1); *try*(RL_2)
Punto de decisión paralelo 	Unión $E_1 \mid E_2$
Actividad Ad hoc 	
	Estrategias de iteración E^* o E^+ o $E!$
Eventos 	Envío de mensajes entre objetos

APÉNDICE: Implementación del ejemplo en Maude

Módulo de sistema

En este módulo se define la estructura del portal, formada por revistas, fascículos y artículos y todas las reglas para su catalogación y consulta.

```
(omod PREU is

      pr NAT .
      pr STRING .

      pr SET{String} .
      pr SET{Nat} .
      pr SET{Tuple{String, String}} .

      pr MAP{String, String} .
      pr MAP{Nat, String} .
      pr MAP{Nat, Tuple{String, Nat}} .

      pr SORTABLE-LIST{String<} .

      class Revista | codigo : String,
                      issn : String,
                      nombre : String,
                      centro : String,
                      tembargo : Nat,
                      editor : String,
                      departamento : String,
                      email : String,
                      idioma : String,
                      periodicidad : Nat,
                      historica : Bool,
                      descripcion : String,
                      materia1 : Nat,
                      listaFasciculos : Set{String},
                      listaSuscripciones : Set{String},
                      indiceTitulos : List{String<},
                      indiceAutores : List{String<} .

      class Fasciculo | codigo : String,
                        fasciculo : String,
                        anno : String,
                        volumen : String,
                        numero : String,
                        fpublicacion : Nat,
                        embargo : Bool,
                        finalizado : Bool,
                        listaArticulos : Set{String} .

      class Anejo | titulo : String .
      subclass Anejo < Fasciculo .

      class Articulo | codigo : String,
                       fasciculo : String,
                       fichero : String,
                       paginaI : String,
```

```
                                paginaF : String,
                                titulo : String,
                                descriptores : String,
                                resumen : String,
                                clase : Nat,
                                listaMaterias2 : Set{Nat},
                                listaAutores : Set{Tuple{String,
String}} .

      class Portal | listaRevistas : Set{String},
                           listaCentros : Map{String, String},
                           listaAreas : Map{Nat, String},
                           listaMaterias1 : Map{Nat, Tuple{String,
Nat}},
                           listaMaterias2 : Map{Nat, Tuple{String,
Nat}},
                           listaClases : Map{Nat, String} .

      class Resultado | rdo : List{String<} .

      var ListaRevistas : Set{String} .
      var ListaCentros : Map{String, String } .
      var Por : Oid .
      vars Codigo ISSN Nombre Centro : String .

      op rev : -> Oid [ctor] .

      msg newRev : String String String String -> Msg .
      msg error : String -> Msg .

      crl [nuevaRevista] :
            newRev(Codigo, ISSN, Nombre, Centro)
            < Por : Portal | listaRevistas : ListaRevistas,
                             listaCentros : ListaCentros >
            =>
            < Por : Portal | listaRevistas : (Codigo, ListaRevistas) >
            < rev : Revista | codigo : Codigo,
                              issn : ISSN,
                              nombre : Nombre,
                              centro : Centro,
                              tembargo : 0,
                              editor : "",
                              departamento : "",
                              email : "",
                              idioma : "",
                              periodicidad : 0,
                              historica : false,
                              descripcion : "",
                              materia1 : 0,
                              listaFasciculos : empty,
                              listaSuscripciones : empty,
                              indiceTitulos : nil,
                              indiceAutores : nil >
            if not Codigo in ListaRevistas
               and length(Codigo) == 4
               and Nombre =/= ""
               and ListaCentros[Centro] =/= undefined .
                  *** validacion de los datos: comprueba
                  ***         que no exista el codigo
                  ***         que su tamaño sea correcto
                  ***         y que el nombre no este vacio
```

```
      crl [nuevaRevista] :
            newRev(Codigo, ISSN, Nombre, Centro)
            < Por : Portal | listaRevistas : ListaRevistas,
                             listaCentros : ListaCentros >
            =>
            error("Error de validacion para la nueva revista " + Codigo
)
            < Por : Portal | >
            if Codigo in ListaRevistas
               or length(Codigo) =/= 4
               or Nombre == ""
               or ListaCentros[Centro] == undefined .

      vars FPublicacion TEmbargo FActual : Nat .

      msg embargo : String Nat -> Msg .

      rl [actualizarEmbargos] :
            embargo(Fasc, FActual)
            < Fas : Fasciculo | codigo : Codigo,
                                fasciculo : Fasc,
                                fpublicacion : FPublicacion >

            < Rev : Revista | codigo : Codigo,
                              tembargo : TEmbargo >
            =>
            < Fas : Fasciculo | embargo : ((FPublicacion + (TEmbargo *
100)) > FActual) >
            < Rev : Revista | > .

      msg actEmbargosFas : Nat Set{String} -> Msg .

      crl [actualizarEmbargos] :
            actEmbargosFas(FActual, (Fasc, ListaFasciculos))
            =>
            embargo(Fasc, FActual)
            actEmbargosFas(FActual, ListaFasciculos)
            if ListaFasciculos =/= empty .

      rl [actualizarEmbargos] :
            actEmbargosFas(FActual, (Fasc, empty))
            =>
            embargo(Fasc, FActual) .

      msg actEmbargosRev : Nat Set{String} -> Msg .

      crl [actualizarEmbargos] :
            actEmbargosRev(FActual, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              listaFasciculos : ListaFasciculos
>
            =>
            < Rev : Revista | >
            actEmbargosFas(FActual, ListaFasciculos)
            actEmbargosRev(FActual, ListaRevistas)
            if ListaRevistas =/= empty and ListaFasciculos =/= empty .

      crl [actualizarEmbargos] :
            actEmbargosRev(FActual, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
```

```
                                    listaFasciculos : empty >
            =>
            < Rev : Revista | >
            actEmbargosRev(FActual, ListaRevistas)
            if ListaRevistas =/= empty .

      crl [actualizarEmbargos] :
            actEmbargosRev(FActual, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                                    listaFasciculos : ListaFasciculos
>
            =>
            < Rev : Revista | >
            actEmbargosFas(FActual, ListaFasciculos)
            if ListaFasciculos =/= empty .

      rl [actualizarEmbargos] :
            actEmbargosRev(FActual, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                                    listaFasciculos : empty >
            =>
            < Rev : Revista | > .

      msg actEmbargos : Nat -> Msg .

      crl [actualizarEmbargos] :
            actEmbargos(FActual)
            < Por : Portal | listaRevistas : ListaRevistas >
            =>
            < Por : Portal | >
            actEmbargosRev(FActual, ListaRevistas)
            if ListaRevistas =/= empty .

      rl [actualizarEmbargos] :
            actEmbargos(FActual)
            < Por : Portal | listaRevistas : empty >
            =>
            < Por : Portal | > .

    vars I B : Nat .
    var A : String .
    var  M : Map{Nat, Tuple{String, Nat}} .
    op imMaterias : Map{Nat, Tuple{String, Nat}} -> Set{String} .
    eq imMaterias(empty) = empty .
    eq imMaterias((I |-> (A, B), M)) = A, imMaterias(M) .

      var Rev : Oid .
      vars Editor Departamento Email Idioma Descripcion Materia1 :
String .
      vars Periodicidad OldTEmbargo : Nat .
      var Historica : Bool .
      vars IMateria1 IArea : Nat .
      var ListaMaterias1 : Map{Nat, Tuple{String, Nat}} .
      msg catRev : String Nat String String String String Nat Bool
String String Nat -> Msg .

      rl [catalogarRevista] :
            catRev(Codigo, TEmbargo, Editor, Departamento, Email,
Idioma, Periodicidad, Historica, Descripcion, Materia1, FActual)
            < Rev : Revista | codigo : Codigo,
                                    tembargo : TEmbargo >
```

```
            < Por : Portal | listaMaterias1 : (IMateria1 |-> (Materia1,
IArea), ListaMaterias1) >
            =>
            < Por : Portal | >
            < Rev : Revista | editor : Editor,
                              departamento : Departamento,
                              email : Email,
                              idioma : Idioma,
                              periodicidad : Periodicidad,
                              historica : Historica,
                              descripcion : Descripcion,
                              materia1 : IMateria1 > .

      crl [catalogarRevista] :
            catRev(Codigo, TEmbargo, Editor, Departamento, Email,
Idioma, Periodicidad, Historica, Descripcion, Materia1, FActual)
            < Rev : Revista | codigo : Codigo,
                              tembargo : OldTEmbargo,
                              listaFasciculos : ListaFasciculos
>
            < Por : Portal | listaMaterias1 : (IMateria1 |-> (Materia1,
IArea), ListaMaterias1) >
            =>
            actEmbargosFas(FActual, ListaFasciculos)
            < Por : Portal | >
            < Rev : Revista | tembargo : TEmbargo,
                              editor : Editor,
                              departamento : Departamento,
                              email : Email,
                              idioma : Idioma,
                              periodicidad : Periodicidad,
                              historica : Historica,
                              descripcion : Descripcion,
                              materia1 : IMateria1 >
            if OldTEmbargo =/= TEmbargo and ListaFasciculos =/= empty .

      crl [catalogarRevista] :
            catRev(Codigo, TEmbargo, Editor, Departamento, Email,
Idioma, Periodicidad, Historica, Descripcion, Materia1, FActual)
            < Rev : Revista | codigo : Codigo,
                              tembargo : OldTEmbargo,
                              listaFasciculos : empty >
            < Por : Portal | listaMaterias1 : (IMateria1 |-> (Materia1,
IArea), ListaMaterias1) >
            =>
            < Por : Portal | >
            < Rev : Revista | tembargo : TEmbargo,
                              editor : Editor,
                              departamento : Departamento,
                              email : Email,
                              idioma : Idioma,
                              periodicidad : Periodicidad,
                              historica : Historica,
                              descripcion : Descripcion,
                              materia1 : IMateria1 >
            if OldTEmbargo =/= TEmbargo .

      crl [catalogarRevista] :
            catRev(Codigo, TEmbargo, Editor, Departamento, Email,
Idioma, Periodicidad, Historica, Descripcion, Materia1, FActual)
            < Por : Portal | listaRevistas : ListaRevistas,
```

```
                                        listaMaterias1 : ListaMaterias1 >
            =>
            error("No se puede actualizar la revista " + Codigo )
            < Por : Portal | >
            if not Codigo in ListaRevistas or not Materia1 in
imMaterias(ListaMaterias1) .

      var Correo : String .
      var ListaSuscripciones : Set{String} .

      msg suscripcion : String String -> Msg .

      crl [suscribirse] :
            suscripcion(Codigo, Correo)
            < Rev : Revista | codigo : Codigo,
                                        listaSuscripciones :
ListaSuscripciones >
            =>
            < Rev : Revista | listaSuscripciones : (Correo,
ListaSuscripciones) >
            if find(Correo, "@", 0) > 0 .

      crl [suscribirse] :
            suscripcion(Codigo, Correo)
            < Por : Portal | listaRevistas : ListaRevistas >
            =>
            error("No se puede suscribir a la revista " + Codigo )
            < Por : Portal | >
            if not Codigo in ListaRevistas or not find(Correo, "@", 0)
> 0 .

      var Fas : Oid .
      vars Anno Volumen Numero Fasc PaginaI PaginaF Fichero : String .
      vars ListaFasciculos ListaArticulos : Set{String} .

      msg carArt : String String String String String String -> Msg .

      ops fas art : -> Oid [ctor] .

*** Caso 1: Se da de alta un artículo en un fasciculo existente
      crl [cargarArticulos] :
            carArt(Codigo, Anno, Volumen, Numero, PaginaI, PaginaF)
            < Fas : Fasciculo | codigo : Codigo,
                                        fasciculo : Fasc,
                                        listaArticulos :
ListaArticulos >
            =>
            < Fas : Fasciculo | listaArticulos : (Fichero,
ListaArticulos) >
            < art : Articulo | codigo : Codigo,
                                    fasciculo : Fasc,
                                    fichero : Fichero,
                                    paginaI : PaginaI,
                                    paginaF : PaginaF,
                                    titulo : "",
                                    descriptores : "",
                                    resumen : "",
                                    clase : 1,
                                    listaMaterias2 :
(empty).Set{Nat},
                                    listaAutores : empty >
```

```
            if Fichero := Codigo + Anno + Volumen + Numero + PaginaI
               /\ Fasc == Codigo + Anno + Volumen + Numero
               /\ not Fichero in ListaArticulos
               /\ length(Anno) == 4 /\ length(Volumen) == 4 /\
length(Numero) == 4
               /\ length(PaginaI) == 4 /\ length(PaginaF) == 4 .

*** Caso 2: El artículo ya existe en el fascículo
      crl [cargarArticulos] :
            carArt(Codigo, Anno, Volumen, Numero, PaginaI, PaginaF)
            < Fas : Fasciculo | codigo : Codigo,
                                fasciculo : Fasc,
                                listaArticulos :
ListaArticulos >
            =>
            error("Ya existe el articulo " + Fichero)
            < Fas : Fasciculo | >
            if Fichero := Codigo + Anno + Volumen + Numero + PaginaI
               /\ Fasc == Codigo + Anno + Volumen + Numero
               /\ Fichero in ListaArticulos .

*** Caso 3: Se da de alta un artículo en un fasciculo nuevo
      crl [cargarArticulos] :
            carArt(Codigo, Anno, Volumen, Numero, PaginaI, PaginaF)
            < Rev : Revista | codigo : Codigo,
                              listaFasciculos : ListaFasciculos
>
            =>
            < Rev : Revista | listaFasciculos : (Fasc, ListaFasciculos)
>
            < fas : Fasciculo | codigo : Codigo,
                                fasciculo : Fasc,
                                anno : Anno,
                                volumen : Volumen,
                                numero : Numero,
                                fpublicacion : 0,
                                embargo : true,
                                finalizado : false,
                                listaArticulos : (Fichero) >

            < art : Articulo | codigo : Codigo,
                               fasciculo : Fasc,
                               fichero : Fichero,
                               paginaI : PaginaI,
                               paginaF : PaginaF,
                               titulo : "",
                               descriptores : "",
                               resumen : "",
                               clase : 1,
                               listaMaterias2 :
(empty).Set{Nat},
                               listaAutores : empty >
            if Fasc := Codigo + Anno + Volumen + Numero
               /\ not Fasc in ListaFasciculos
               /\ Fichero := Codigo + Anno + Volumen + Numero + PaginaI
               /\ length(Anno) == 4 /\ length(Volumen) == 4 /\
length(Numero) == 4
               /\ length(PaginaI) == 4 /\ length(PaginaF) == 4 .

*** Caso 4: No existe la revista en donde se quiere dar de alta el
articulo
```

```
      crl [cargarArticulos] :
            carArt(Codigo, Anno, Volumen, Numero, PaginaI, PaginaF)
            < Por : Portal | listaRevistas : ListaRevistas >
            =>
            error("No existe la revista " + Codigo)
            < Por : Portal | >
            if not Codigo in ListaRevistas .

*** Caso 5: Error en la validacion de datos
      crl [cargarArticulos] :
            carArt(Codigo, Anno, Volumen, Numero, PaginaI, PaginaF)
            =>
            error("Error de validacion para el nuevo articulo " +
Fichero )
            if Fichero := Codigo + Anno + Volumen + Numero + PaginaI
               /\ (length(Anno) =/= 4 or length(Volumen) =/= 4 or
length(Numero) =/= 4
               or length(PaginaI) =/= 4 or length(PaginaF) =/= 4) .

      var C : Map{Nat, String} .
    op imClases : Map{Nat, String} -> Set{String} .
    eq imClases(empty) = empty .
    eq imClases((I |-> A, C)) = A, imClases(C) .

      var Materia : String .
      vars IMateria IMat : Nat .
      var RestoMaterias : Set{String} .
      var ListaMaterias : Map{Nat, Tuple{String, Nat}} .
    op materiasEn : Set{String} Map{Nat, Tuple{String, Nat}} ->
Set{Nat} .
    eq materiasEn(empty, ListaMaterias) = empty .
    eq materiasEn((Materia, RestoMaterias), (IMateria |-> (Materia,
IMat), ListaMaterias)) = IMateria, materiasEn(RestoMaterias, (IMateria
|-> (Materia, IMat), ListaMaterias)) .
    ceq materiasEn((Materia, RestoMaterias), ListaMaterias) =
materiasEn(RestoMaterias, ListaMaterias)
      if not Materia in imMaterias(ListaMaterias) .

      var Art : Oid .
      vars PagI PagF Titulo Descriptores Resumen Clase OldFichero :
String .
      vars IClase IMateria2 : Nat .
      var Materias2 : Set{String} .
      var ListaMaterias2 : Map{Nat, Tuple{String, Nat}} .
      vars ListaMat2 NewMaterias : Set{Nat} .
      var ListaClases : Map{Nat, String} .
      vars Autores ListaAutores : Set{Tuple{String, String}} .

      msg catArt : String String String String String String String
String String String Set{String} Set{Tuple{String, String}} -> Msg .

      crl [catalogarArticulo] :
            catArt(Codigo, Fasc, Fichero, PaginaI, PaginaF, Titulo,
Descriptores, Resumen, Clase, Materias2, Autores)
            < Art : Articulo | codigo : Codigo,
                               fasciculo : Fasc,
                               fichero : Fichero,
                               paginaI : PaginaI,
                               listaMaterias2 : ListaMat2,
                               listaAutores : ListaAutores >
            < Por : Portal | listaMaterias2 : ListaMaterias2,
```

```
                                    listaClases : (IClase |-> Clase,
ListaClases) >
            =>
            < Por : Portal | >
            < Art : Articulo | paginaF : PaginaF,
                               titulo : Titulo,
                               descriptores : Descriptores,
                               resumen : Resumen,
                               clase : IClase,
                               listaMaterias2 : NewMaterias,
ListaMat2,
                               listaAutores : Autores,
ListaAutores >
            if NewMaterias := materiasEn(Materias2, ListaMaterias2)
            /\ | NewMaterias | == | Materias2 | .

      crl [catalogarArticulo] :
            catArt(Codigo, Fasc, OldFichero, PaginaI, PaginaF, Titulo,
Descriptores, Resumen, Clase, Materias2, Autores)
            < Art : Articulo | codigo : Codigo,
                               fasciculo : Fasc,
                               fichero : OldFichero,
                               paginaI : PagI,
                               listaMaterias2 : ListaMat2,
                               listaAutores : ListaAutores >
            < Fas : Fasciculo | listaArticulos : (OldFichero,
ListaArticulos) >
            < Por : Portal | listaMaterias2 : ListaMaterias2,
                             listaClases : (IClase |-> Clase,
ListaClases) >
            =>
            < Por : Portal | >
            < Fas : Fasciculo | listaArticulos : (Fichero,
ListaArticulos) >
            < Art : Articulo | fichero : Fichero,
                               paginaI : PaginaI,
                               paginaF : PaginaF,
                               titulo : Titulo,
                               descriptores : Descriptores,
                               resumen : Resumen,
                               clase : IClase,
                               listaMaterias2 : NewMaterias,
ListaMat2,
                               listaAutores : Autores,
ListaAutores >
            if Fichero := Fasc + PaginaI /\ PaginaI =/= PagI
            /\ NewMaterias := materiasEn(Materias2, ListaMaterias2)
            /\ | NewMaterias | == | Materias2 |
            /\ not Fichero in ListaArticulos .

      crl [catalogarArticulo] :
            catArt(Codigo, Fasc, Fichero, PaginaI, PaginaF, Titulo,
Descriptores, Resumen, Clase, Materias2, Autores)
            < Art : Articulo | codigo : Codigo,
                               fasciculo : Fasc,
                               fichero : Fichero,
                               paginaI : PaginaI,
                               listaMaterias2 : ListaMat2,
                               listaAutores : ListaAutores >
            < Por : Portal | listaMaterias2 : ListaMaterias2,
```

```
                                    listaClases : (IClase |-> Clase,
ListaClases) >
            =>
            < Por : Portal | >
            < Art : Articulo | paginaF : PaginaF,
                               titulo : Titulo,
                               descriptores : Descriptores,
                               resumen : Resumen,
                               clase : IClase,
                               listaMaterias2 : NewMaterias,
ListaMat2,
                               listaAutores : Autores,
ListaAutores >
            if NewMaterias := materiasEn(Materias2, ListaMaterias2)
            /\ | NewMaterias | == | Materias2 | .

      crl [catalogarArticulo] :
            catArt(Codigo, Fasc, Fichero, PaginaI, PaginaF, Titulo,
Descriptores, Resumen, Clase, Materias2, Autores)
            < Fas : Fasciculo | codigo : Codigo,
                                fasciculo : Fasc,
                                listaArticulos :
ListaArticulos >
            < Por : Portal | listaMaterias2 : ListaMaterias2,
                             listaClases : ListaClases >
            =>
            error("No se puede actualizar el articulo " + Fichero )
            < Por : Portal | >
            < Fas : Fasciculo | >
            if not Fichero in ListaArticulos
            or not | materiasEn(Materias2, ListaMaterias2) | == |
Materias2 |
            or not Clase in imClases(ListaClases) .

      crl [catalogarArticulo] :
            catArt(Codigo, Fasc, Fichero, PaginaI, PaginaF, Titulo,
Descriptores, Resumen, Clase, Materias2, Autores)
            < Rev : Revista | codigo : Codigo,
                              listaFasciculos : ListaFasciculos
>
            =>
            error("No se puede actualizar el articulo " + Fichero + ".
No existe el fasciculo " + Fasc )
            < Rev : Revista | >
            if not Fasc in ListaFasciculos .

      crl [catalogarArticulo] :
            catArt(Codigo, Fasc, Fichero, PaginaI, PaginaF, Titulo,
Descriptores, Resumen, Clase, Materias2, Autores)
            < Por : Portal | listaRevistas : ListaRevistas >
            =>
            error("No se puede actualizar el articulo " + Fichero + ".
No existe la revista " + Codigo )
            < Por : Portal | >
            if not Codigo in ListaRevistas .

      msg alerta : String -> Msg .

      rl [enviarAlerta] :
            alerta(Fasc)
            < Rev : Revista | codigo : Codigo,
```

```
                                        listaSuscripciones :
ListaSuscripciones >
            < Fas : Fasciculo | codigo : Codigo,
                                          fasciculo : Fasc >
            =>
            < Rev : Revista | >
            < Fas : Fasciculo | finalizado : true > .

      var FicheroOld : String .
      var ListaArt : Set{String} .

      msg cambioFichero : String Set{String} -> Msg .

      crl [catalogarFasciculo] :
            cambioFichero(Fasc, (FicheroOld, ListaArticulos))
            < Art : Articulo | fichero : FicheroOld,
                                       paginaI : PaginaI >
            < Fas : Fasciculo | fasciculo : Fasc,
                                        listaArticulos :
(FicheroOld, ListaArt) >
            =>
            cambioFichero(Fasc, ListaArticulos)
            < Art : Articulo | fasciculo : Fasc,
                                       fichero : Fichero >
            < Fas : Fasciculo | listaArticulos : (Fichero, ListaArt) >
            if Fichero := Fasc + PaginaI /\ ListaArticulos =/= empty .

      crl [catalogarFasciculo] :
            cambioFichero(Fasc, (FicheroOld, empty))
            < Art : Articulo | fichero : FicheroOld,
                                       paginaI : PaginaI >
            < Fas : Fasciculo | fasciculo : Fasc,
                                        listaArticulos :
(FicheroOld, ListaArt) >
            =>
            < Art : Articulo | fasciculo : Fasc,
                                       fichero : Fichero >
            < Fas : Fasciculo | listaArticulos : (Fichero, ListaArt) >
            if Fichero := Fasc + PaginaI .

      vars Ann Vol Num : String .
      msg catFas : String String String String String Nat Nat -> Msg .

***   Caso 1: No hay cambios en el nombre del fasciculo
      rl [catalogarFasciculo] :
            catFas(Codigo, Fasc, Anno, Volumen, Numero, FPublicacion,
FActual)
            < Fas : Fasciculo | codigo : Codigo,
                                          fasciculo : Fasc,
                                          anno : Anno,
                                          volumen : Volumen,
                                          numero : Numero >
            =>
            embargo(Fasc, FActual)
            alerta(Fasc)
            < Fas : Fasciculo | fpublicacion : FPublicacion > .

***   Caso 2: Hay cambios en el nombre del fasciculo, hay que
actualizar los articulos
      crl [catalogarFasciculo] :
```

```
            catFas(Codigo, Fasc, Anno, Volumen, Numero, FPublicacion,
FActual)
            < Rev : Revista | codigo : Codigo,
                                    listaFasciculos : (Fasc,
ListaFasciculos) >
            < Fas : Fasciculo | codigo : Codigo,
                                    fasciculo : Fasc,
                                    anno : Ann,
                                    volumen : Vol,
                                    numero : Num,
                                    listaArticulos :
ListaArticulos >
            =>
            embargo(Codigo + Anno + Volumen + Numero, FActual)
            cambioFichero(Codigo + Anno + Volumen + Numero,
ListaArticulos)
            alerta(Codigo + Anno + Volumen + Numero)
            < Rev : Revista | codigo : Codigo,
                                    listaFasciculos : (Codigo + Anno
+ Volumen + Numero, ListaFasciculos) >
            < Fas : Fasciculo | fasciculo : Codigo + Anno + Volumen +
Numero,
                                    anno : Anno,
                                    volumen : Volumen,
                                    numero : Numero,
                                    fpublicacion : FPublicacion >
            if (Anno =/= Ann or Volumen =/= Vol or Numero =/= Num)
            and length(Anno) == 4 and length(Volumen) == 4 and
length(Numero) == 4
            and not (Codigo + Anno + Volumen + Numero) in
ListaFasciculos .

***   Casos 3, 4, 5 y 6: casos de error
      crl [catalogarFasciculo] :
            catFas(Codigo, Fasc, Anno, Volumen, Numero, FPublicacion,
FActual)
            < Rev : Revista | codigo : Codigo,
                                    listaFasciculos : (Fasc,
ListaFasciculos) >
            < Fas : Fasciculo | codigo : Codigo,
                                    fasciculo : Fasc,
                                    anno : Ann,
                                    volumen : Vol,
                                    numero : Num >
            =>
            error("Error de validacion para el fasciculo " + Fasc )
            < Rev : Revista | >
            < Fas : Fasciculo | >
            if length(Anno) =/= 4 or length(Volumen) =/= 4 or
length(Numero) =/= 4
            or (Codigo + Anno + Volumen + Numero) in ListaFasciculos .

      crl [catalogarFasciculo] :
            catFas(Codigo, Fasc, Anno, Volumen, Numero, FPublicacion,
FActual)
            < Rev : Revista | codigo : Codigo,
                                    listaFasciculos : ListaFasciculos
>
            =>
            error("No se puede actualizar el fasciculo " + Fasc + ". No
existe." )
```

```
            < Rev : Revista | >
            if not Fasc in ListaFasciculos .

      crl [catalogarFasciculo] :
            catFas(Codigo, Fasc, Anno, Volumen, Numero, FPublicacion,
FActual)
            < Por : Portal | listaRevistas : ListaRevistas >
            =>
            error("No se puede actualizar el fasciculo " + Fasc + ". No
existe la revista " + Codigo )
            < Por : Portal | >
            if not Codigo in ListaRevistas .

      vars IndiceTitulos IndTit : List{String<} .

      msg indTitulosArt : Set{String} -> Msg .

      crl [generarIndiceTitulos] :
            indTitulosArt((Fichero, ListaArticulos))
            < Art : Articulo | codigo : Codigo,
                               fichero : Fichero,
                               titulo : Titulo >

            < Rev : Revista | codigo : Codigo,
                              indiceTitulos : IndiceTitulos >
            =>
            < Rev : Revista | indiceTitulos : IndTit >
            < Art : Articulo | >
            indTitulosArt(ListaArticulos)
            if IndTit := sort((Titulo IndiceTitulos))
            /\ Titulo =/= "" /\ ListaArticulos =/= empty .

      crl [generarIndiceTitulos] :
            indTitulosArt((Fichero, ListaArticulos))
            < Art : Articulo | codigo : Codigo,
                               fichero : Fichero,
                               titulo : "" >

            < Rev : Revista | codigo : Codigo,
                              indiceTitulos : IndiceTitulos >
            =>
            < Rev : Revista | indiceTitulos : IndiceTitulos >
            < Art : Articulo | >
            indTitulosArt(ListaArticulos)
            if ListaArticulos =/= empty .

      crl [generarIndiceTitulos] :
            indTitulosArt((Fichero, empty))
            < Art : Articulo | codigo : Codigo,
                               fichero : Fichero,
                               titulo : Titulo >

            < Rev : Revista | codigo : Codigo,
                              indiceTitulos : IndiceTitulos >
            =>
            < Rev : Revista | indiceTitulos : IndTit >
            < Art : Articulo | >
            if IndTit := sort((Titulo IndiceTitulos)) /\ Titulo =/= ""
.

      rl [generarIndiceTitulos] :
```

```
            indTitulosArt((Fichero, empty))
            < Art : Articulo | codigo : Codigo,
                               fichero : Fichero,
                               titulo : "" >

            < Rev : Revista | codigo : Codigo,
                              indiceTitulos : IndiceTitulos >
            =>
            < Rev : Revista | indiceTitulos : IndiceTitulos >
            < Art : Articulo | > .

      msg indTitulosFas : Set{String} -> Msg .

      crl [generarIndiceTitulos] :
            indTitulosFas((Fasc, ListaFasciculos))
            < Fas : Fasciculo | fasciculo : Fasc,
                                listaArticulos :
ListaArticulos >
            =>
            < Fas : Fasciculo | >
            indTitulosArt(ListaArticulos)
            indTitulosFas(ListaFasciculos)
            if ListaFasciculos =/= empty and ListaArticulos =/= empty .

      crl [generarIndiceTitulos] :
            indTitulosFas((Fasc, ListaFasciculos))
            < Fas : Fasciculo | fasciculo : Fasc,
                                listaArticulos : empty >
            =>
            < Fas : Fasciculo | >
            indTitulosFas(ListaFasciculos)
            if ListaFasciculos =/= empty .

      crl [generarIndiceTitulos] :
            indTitulosFas((Fasc, empty))
            < Fas : Fasciculo | fasciculo : Fasc,
                                listaArticulos :
ListaArticulos >
            =>
            < Fas : Fasciculo | >
            indTitulosArt(ListaArticulos)
            if ListaArticulos =/= empty .

      rl [generarIndiceTitulos] :
            indTitulosFas((Fasc, empty))
            < Fas : Fasciculo | fasciculo : Fasc,
                                listaArticulos : empty >
            =>
            < Fas : Fasciculo | > .

      msg indTitulosRev : Set{String} -> Msg .

      crl [generarIndiceTitulos] :
            indTitulosRev((Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              listaFasciculos :
ListaFasciculos,
                              indiceTitulos : IndiceTitulos >
            =>
            < Rev : Revista | indiceTitulos : nil >
            indTitulosFas(ListaFasciculos)
```

```
            indTitulosRev(ListaRevistas)
            if ListaRevistas =/= empty and ListaFasciculos =/= empty .

      crl [generarIndiceTitulos] :
            indTitulosRev((Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              listaFasciculos : empty >
            =>
            < Rev : Revista | >
            indTitulosRev(ListaRevistas)
            if ListaRevistas =/= empty .

      crl [generarIndiceTitulos] :
            indTitulosRev((Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                              listaFasciculos :
ListaFasciculos,
                              indiceTitulos : IndiceTitulos >
            =>
            < Rev : Revista | indiceTitulos : nil >
            indTitulosFas(ListaFasciculos)
            if ListaFasciculos =/= empty .

      rl [generarIndiceTitulos] :
            indTitulosRev((Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                              listaFasciculos : empty >
            =>
            < Rev : Revista | > .

      op indTitulos : -> Msg .

      crl [generarIndiceTitulos] :
            indTitulos
            < Por : Portal | listaRevistas : ListaRevistas >
            =>
            < Por : Portal | >
            indTitulosRev(ListaRevistas)
            if ListaRevistas =/= empty .

      rl [generarIndiceTitulos] :
            indTitulos
            < Por : Portal | listaRevistas : empty >
            =>
            < Por : Portal | > .

      var IndiceAutores : List{String<} .
      vars Apellidos Nombre : String .

      op autores : Tuple{String, String} List{String<} ->
List{String<} .
      eq autores((Apellidos, Nombre), IndiceAutores) = (Apellidos + ",
" + Nombre) IndiceAutores .

      var Autor : Tuple{String, String} .

      op autoresLis : Set{Tuple{String, String}} List{String<} ->
List{String<} .
      eq autoresLis(empty, IndiceAutores) = sort(IndiceAutores) .
      eq autoresLis((Autor, ListaAutores), IndiceAutores) =
autoresLis(ListaAutores, autores(Autor, IndiceAutores)) [owise] .
```

```
      var IndAut : List{String<} .

      msg indAutoresArt : Set{String} -> Msg .

      crl [generarIndiceAutores] :
            indAutoresArt((Fichero, ListaArticulos))
            < Art : Articulo | codigo : Codigo,
                               fichero : Fichero,
                               listaAutores : ListaAutores >

            < Rev : Revista | codigo : Codigo,
                              indiceAutores : IndiceAutores >
            =>
            < Rev : Revista | indiceAutores : IndAut >
            < Art : Articulo | >
            indAutoresArt(ListaArticulos)
            if IndAut := autoresLis(ListaAutores, IndiceAutores)
            /\ ListaAutores =/= empty /\ ListaArticulos =/= empty .

      crl [generarIndiceAutores] :
            indAutoresArt((Fichero, ListaArticulos))
            < Art : Articulo | codigo : Codigo,
                               fichero : Fichero,
                               listaAutores : empty >

            =>
            < Art : Articulo | >
            indAutoresArt(ListaArticulos)
            if ListaArticulos =/= empty .

      crl [generarIndiceAutores] :
            indAutoresArt((Fichero, empty))
            < Art : Articulo | codigo : Codigo,
                               fichero : Fichero,
                               listaAutores : ListaAutores >

            < Rev : Revista | codigo : Codigo,
                              indiceAutores : IndiceAutores >
            =>
            < Rev : Revista | indiceAutores : IndAut >
            < Art : Articulo | >
            if IndAut := autoresLis(ListaAutores, IndiceAutores)
            /\ ListaAutores =/= empty .

      rl [generarIndiceAutores] :
            indAutoresArt((Fichero, empty))
            < Art : Articulo | codigo : Codigo,
                               fichero : Fichero,
                               listaAutores : empty >

            =>
            < Art : Articulo | > .

      msg indAutoresFas : Set{String} -> Msg .

      crl [generarIndiceAutores] :
            indAutoresFas((Fasc, ListaFasciculos))
            < Fas : Fasciculo | fasciculo : Fasc,
                                listaArticulos :
ListaArticulos >
```

```
            =>
            < Fas : Fasciculo | >
            indAutoresArt(ListaArticulos)
            indAutoresFas(ListaFasciculos)
            if ListaFasciculos =/= empty and ListaArticulos =/= empty .

      crl [generarIndiceAutores] :
            indAutoresFas((Fasc, ListaFasciculos))
            < Fas : Fasciculo | fasciculo : Fasc,
                                    listaArticulos : empty >
            =>
            < Fas : Fasciculo | >
            indAutoresFas(ListaFasciculos)
            if ListaFasciculos =/= empty .

      crl [generarIndiceAutores] :
            indAutoresFas((Fasc, empty))
            < Fas : Fasciculo | fasciculo : Fasc,
                                    listaArticulos :
ListaArticulos >
            =>
            < Fas : Fasciculo | >
            indAutoresArt(ListaArticulos)
            if ListaArticulos =/= empty .

      rl [generarIndiceAutores] :
            indAutoresFas((Fasc, empty))
            < Fas : Fasciculo | fasciculo : Fasc,
                                    listaArticulos : empty >
            =>
            < Fas : Fasciculo | > .

      msg indAutoresRev : Set{String} -> Msg .

      crl [generarIndiceAutores] :
            indAutoresRev((Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                                listaFasciculos :
ListaFasciculos,
                                indiceAutores : IndiceAutores >
            =>
            < Rev : Revista | indiceAutores : nil >
            indAutoresFas(ListaFasciculos)
            indAutoresRev(ListaRevistas)
            if ListaRevistas =/= empty and ListaFasciculos =/= empty .

      crl [generarIndiceAutores] :
            indAutoresRev((Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                                listaFasciculos : empty >
            =>
            < Rev : Revista | >
            indAutoresRev(ListaRevistas)
            if ListaRevistas =/= empty .

      crl [generarIndiceAutores] :
            indAutoresRev((Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                                listaFasciculos :
ListaFasciculos,
                                indiceAutores : IndiceAutores >
```

```
      =>
      < Rev : Revista | indiceAutores : nil >
      indAutoresFas(ListaFasciculos)
      if ListaFasciculos =/= empty .

rl [generarIndiceAutores] :
      indAutoresRev((Codigo, empty))
      < Rev : Revista | codigo : Codigo,
                        listaFasciculos : empty >
      =>
      < Rev : Revista | > .

op indAutores : -> Msg .

crl [generarIndiceAutores] :
      indAutores
      < Por : Portal | listaRevistas : ListaRevistas >
      =>
      < Por : Portal | >
      indAutoresRev(ListaRevistas)
      if ListaRevistas =/= empty .

rl [generarIndiceAutores] :
      indAutores
      < Por : Portal | listaRevistas : empty >
      =>
      < Por : Portal | > .

var Termino : String .
var ListaNombres ListaRdo : List{String<} .
op redo : -> Oid [ctor] .

msg indiceTitulos : String -> Msg .

rl [consulta] :
      indiceTitulos(Codigo)
      < Rev : Revista | codigo : Codigo,
                        indiceTitulos : IndiceTitulos >
      =>
      < Rev : Revista | >
      < redo : Resultado | rdo : IndiceTitulos > .

crl [consulta] :
      indiceTitulos(Codigo)
      < Por : Portal | listaRevistas : ListaRevistas >
      =>
      < Por : Portal | >
      < redo : Resultado | rdo : "No hay resultados" >
      if not Codigo in ListaRevistas .

msg indiceAutores : String -> Msg .

rl [consulta] :
      indiceAutores(Codigo)
      < Rev : Revista | codigo : Codigo,
                        indiceAutores : IndiceAutores >
      =>
      < Rev : Revista | >
      < redo : Resultado | rdo : IndiceAutores > .

crl [consulta] :
```

```
            indiceAutores(Codigo)
            < Por : Portal | listaRevistas : ListaRevistas >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : "No hay resultados" >
            if not Codigo in ListaRevistas .

      msg acceder : String String String -> Msg .

      rl [consulta] :
            acceder(Codigo, Fasc, Fichero)
            < Art : Articulo | codigo : Codigo,
                               fasciculo : Fasc,
                               fichero : Fichero >
            < Fas : Fasciculo | codigo : Codigo,
                               fasciculo : Fasc,
                               embargo : false >
            =>
            < Art : Articulo | >
            < redo : Resultado | rdo : "PDF " + Fichero + " descargado"
> .

      rl [consulta] :
            acceder(Codigo, Fasc, Fichero)
            < Art : Articulo | codigo : Codigo,
                               fasciculo : Fasc,
                               fichero : Fichero >
            < Fas : Fasciculo | codigo : Codigo,
                               fasciculo : Fasc,
                               embargo : true >
            =>
            < Art : Articulo | >
            < redo : Resultado | rdo : "PDF " + Fichero + " no
disponible por embargo." > .

      crl [consulta] :
            acceder(Codigo, Fasc, Fichero)
            < Fas : Fasciculo | codigo : Codigo,
                                fasciculo : Fasc,
                                listaArticulos : ListaArticulos
>
            =>
            < Fas : Fasciculo | >
            < redo : Resultado | rdo : "No existe el fichero " +
Fichero >
            if not Fichero in ListaArticulos .

      crl [consulta] :
            acceder(Codigo, Fasc, Fichero)
            < Rev : Revista | codigo : Codigo,
                              listaFasciculos : ListaFasciculos
>
            =>
            < Rev : Revista | >
            < redo : Resultado | rdo : "No existe el fasciculo " + Fasc
>
            if not Fasc in ListaFasciculos .

      crl [consulta] :
            acceder(Codigo, Fasc, Fichero)
            < Por : Portal | listaRevistas : ListaRevistas >
```

```
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : "No existe la revista " + Codigo
>
            if not Codigo in ListaRevistas .

      msg buscarRevLista : String Set{String} -> Msg .

      crl [consulta] :
            buscarRevLista(Termino, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre >
          < redo : Resultado | rdo : ListaNombres >
            =>
            < Rev : Revista | >
            < redo : Resultado | rdo : sort(Nombre ListaNombres) >
            buscarRevLista(Termino, ListaRevistas)
            if ListaRevistas =/= empty
            and find(Nombre, Termino, 0) >= 0 .

      crl [consulta] :
            buscarRevLista(Termino, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre,
                              descripcion : Descripcion >
          < redo : Resultado | rdo : ListaNombres >
            =>
            < Rev : Revista | >
            < redo : Resultado | rdo : sort(Nombre ListaNombres) >
            buscarRevLista(Termino, ListaRevistas)
            if ListaRevistas =/= empty
            and find(Descripcion, Termino, 0) >= 0 .

      crl [consulta] :
            buscarRevLista(Termino, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre,
                              descripcion : Descripcion >
            =>
            < Rev : Revista | >
            buscarRevLista(Termino, ListaRevistas)
            if ListaRevistas =/= empty
            and find(Nombre, Termino, 0) == notFound
            and find(Descripcion, Termino, 0) == notFound .

      crl [consulta] :
            buscarRevLista(Termino, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre >
            < redo : Resultado | rdo : ListaNombres >
            =>
            < Rev : Revista | >
            < redo : Resultado | rdo : sort(Nombre ListaNombres) >
            if find(Nombre, Termino, 0) >= 0 .

      crl [consulta] :
            buscarRevLista(Termino, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre,
                              descripcion : Descripcion >
            < redo : Resultado | rdo : ListaNombres >
```

```
            =>
            < Rev : Revista | >
            < redo : Resultado | rdo : sort(Nombre ListaNombres) >
            if find(Descripcion, Termino, 0) >= 0 .

      crl [consulta] :
            buscarRevLista(Termino, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre,
                              descripcion : Descripcion >
            =>
            < Rev : Revista | >
            if (find(Nombre, Termino, 0) == notFound)
            and (find(Descripcion, Termino, 0) == notFound) .

      msg buscarRev : String -> Msg .

      crl [consulta] :
            buscarRev(Termino)
            < Por : Portal | listaRevistas : ListaRevistas >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : nil >
            buscarRevLista(Termino, ListaRevistas)
            if ListaRevistas =/= empty .

      rl [consulta] :
            buscarRev(Termino)
            < Por : Portal | listaRevistas : empty >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : "No hay resultados" > .

      var ListaTitulos : List{String<} .

      msg buscarArtAutores : String String Set{Tuple{String, String}}
-> Msg .

      crl [consulta] :
            buscarArtAutores(Termino, Titulo, ((Apellidos, Nombre),
ListaAutores))
            < redo : Resultado | rdo : ListaTitulos >
            =>
            < redo : Resultado | rdo : sort(Titulo ListaTitulos) >
            if find(Nombre, Termino, 0) >= 0 .

      crl [consulta] :
            buscarArtAutores(Termino, Titulo, ((Apellidos, Nombre),
ListaAutores))
            < redo : Resultado | rdo : ListaTitulos >
            =>
            < redo : Resultado | rdo : sort(Titulo ListaTitulos) >
            if find(Apellidos, Termino, 0) >= 0 .

      crl [consulta] :
            buscarArtAutores(Termino, Titulo, ((Apellidos, Nombre),
ListaAutores))
            =>
            buscarArtAutores(Termino, Titulo, ListaAutores)
            if find(Apellidos, Termino, 0) == notFound
            or find(Nombre, Termino, 0) == notFound .
```

```
rl [consulta] :
      buscarArtAutores(Termino, Titulo, empty)
      < redo : Resultado | rdo : ListaTitulos >
      =>
      < redo : Resultado | > .

msg buscarArtLista : String Set{String} -> Msg .

crl [consulta] :
      buscarArtLista(Termino, (Fichero, ListaArticulos))
      < Art : Articulo | fichero : Fichero,
                         titulo : Titulo,
                         descriptores : Descriptores,
                         resumen : Resumen >
    < redo : Resultado | rdo : ListaTitulos >
      =>
      < Art : Articulo | >
      < redo : Resultado | rdo : sort(Titulo ListaTitulos) >
      buscarArtLista(Termino, ListaArticulos)
      if ListaArticulos =/= empty
      and find(Titulo, Termino, 0) >= 0 .

crl [consulta] :
      buscarArtLista(Termino, (Fichero, ListaArticulos))
      < Art : Articulo | fichero : Fichero,
                         titulo : Titulo,
                         descriptores : Descriptores,
                         resumen : Resumen >
    < redo : Resultado | rdo : ListaTitulos >
      =>
      < Art : Articulo | >
      < redo : Resultado | rdo : sort(Titulo ListaTitulos) >
      buscarArtLista(Termino, ListaArticulos)
      if ListaArticulos =/= empty
      and find(Descriptores, Termino, 0) >= 0 .

crl [consulta] :
      buscarArtLista(Termino, (Fichero, ListaArticulos))
      < Art : Articulo | fichero : Fichero,
                         titulo : Titulo,
                         descriptores : Descriptores,
                         resumen : Resumen >
    < redo : Resultado | rdo : ListaTitulos >
      =>
      < Art : Articulo | >
      < redo : Resultado | rdo : sort(Titulo ListaTitulos) >
      buscarArtLista(Termino, ListaArticulos)
      if ListaArticulos =/= empty
      and find(Resumen, Termino, 0) >= 0 .

crl [consulta] :
      buscarArtLista(Termino, (Fichero, ListaArticulos))
      < Art : Articulo | fichero : Fichero,
                         titulo : Titulo,
                         descriptores : Descriptores,
                         resumen : Resumen,
                         listaAutores : ListaAutores >
      =>
      < Art : Articulo | >
      buscarArtAutores(Termino, Titulo, ListaAutores)
```

```
            buscarArtLista(Termino, ListaArticulos)
            if ListaArticulos =/= empty
            and find(Titulo, Termino, 0) == notFound
            and find(Descriptores, Termino, 0) == notFound
            and find(Resumen, Termino, 0) == notFound .

      crl [consulta] :
            buscarArtLista(Termino, (Fichero, empty))
            < Art : Articulo | fichero : Fichero,
                               titulo : Titulo >
            < redo : Resultado | rdo : ListaTitulos >
            =>
            < Art : Articulo | >
            < redo : Resultado | rdo : sort(Titulo ListaTitulos) >
            if find(Titulo, Termino, 0) >= 0 .

      crl [consulta] :
            buscarArtLista(Termino, (Fichero, empty))
            < Art : Articulo | fichero : Fichero,
                               titulo : Titulo,
                               descriptores : Descriptores >
            < redo : Resultado | rdo : ListaTitulos >
            =>
            < Art : Articulo | >
            < redo : Resultado | rdo : sort(Titulo ListaTitulos) >
            if find(Descriptores, Termino, 0) >= 0 .

      crl [consulta] :
            buscarArtLista(Termino, (Fichero, empty))
            < Art : Articulo | fichero : Fichero,
                               titulo : Titulo,
                               resumen : Resumen >
            < redo : Resultado | rdo : ListaTitulos >
            =>
            < Art : Articulo | >
            < redo : Resultado | rdo : sort(Titulo ListaTitulos) >
            if find(Resumen, Termino, 0) >= 0 .

      crl [consulta] :
            buscarArtLista(Termino, (Fichero, empty))
            < Art : Articulo | fichero : Fichero,
                               titulo : Titulo,
                               descriptores : Descriptores,
                               resumen : Resumen,
                               listaAutores : ListaAutores >
            =>
            < Art : Articulo | >
            buscarArtAutores(Termino, Titulo, ListaAutores)
            if find(Titulo, Termino, 0) == notFound
            and find(Descriptores, Termino, 0) == notFound
            and find(Resumen, Termino, 0) == notFound .

      msg buscarArtFas : String Set{String} -> Msg .

      crl [consulta] :
            buscarArtFas(Termino, (Fasc, ListaFasciculos))
            < Fas : Fasciculo | fasciculo : Fasc,
                                listaArticulos :
ListaArticulos >
            =>
            < Fas : Fasciculo | >
```

```
            buscarArtLista(Termino, ListaArticulos)
            buscarArtFas(Termino, ListaFasciculos)
            if ListaFasciculos =/= empty and ListaArticulos =/= empty .

      crl [consulta] :
            buscarArtFas(Termino, (Fasc, ListaFasciculos))
            < Fas : Fasciculo | fasciculo : Fasc,
                                      listaArticulos : empty >
            =>
            < Fas : Fasciculo | >
            buscarArtFas(Termino, ListaFasciculos)
            if ListaFasciculos =/= empty .

      crl [consulta] :
            buscarArtFas(Termino, (Fasc, empty))
            < Fas : Fasciculo | fasciculo : Fasc,
                                      listaArticulos :
ListaArticulos >
            =>
            < Fas : Fasciculo | >
            buscarArtLista(Termino, ListaArticulos)
            if ListaArticulos =/= empty .

      rl [consulta] :
            buscarArtFas(Termino, (Fasc, empty))
            < Fas : Fasciculo | fasciculo : Fasc,
                                      listaArticulos : empty >
            =>
            < Fas : Fasciculo | > .

      msg buscarArtRev : String Set{String} -> Msg .

      crl [consulta] :
            buscarArtRev(Termino, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                                 listaFasciculos : ListaFasciculos
>
            =>
            < Rev : Revista | >
            buscarArtFas(Termino, ListaFasciculos)
            buscarArtRev(Termino, ListaRevistas)
            if ListaRevistas =/= empty and ListaFasciculos =/= empty .

      crl [consulta] :
            buscarArtRev(Termino, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                                 listaFasciculos : empty >
            =>
            < Rev : Revista | >
            buscarArtRev(Termino, ListaRevistas)
            if ListaRevistas =/= empty .

      crl [consulta] :
            buscarArtRev(Termino, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                                 listaFasciculos : ListaFasciculos
>
            =>
            < Rev : Revista | >
            buscarArtFas(Termino, ListaFasciculos)
            if ListaFasciculos =/= empty .
```

```
rl [consulta] :
      buscarArtRev(Termino, (Codigo, empty))
      < Rev : Revista | codigo : Codigo,
                        listaFasciculos : empty >
      =>
      < Rev : Revista | > .

msg buscarArt : String -> Msg .

crl [consulta] :
      buscarArt(Termino)
      < Por : Portal | listaRevistas : ListaRevistas >
      =>
      < Por : Portal | >
      < redo : Resultado | rdo : nil >
      buscarArtRev(Termino, ListaRevistas)
      if ListaRevistas =/= empty .

rl [consulta] :
      buscarArt(Termino)
      < Por : Portal | listaRevistas : empty >
      =>
      < Por : Portal | >
      < redo : Resultado | rdo : "No hay resultados" > .

msg listado : Set{String} -> Msg .

crl [consulta] :
      listado((Codigo, ListaRevistas))
      < Rev : Revista | codigo : Codigo,
                        nombre : Nombre >
      < redo : Resultado | rdo : ListaNombres >
      =>
      < Rev : Revista | >
      < redo : Resultado | rdo : sort(Nombre ListaNombres) >
      listado(ListaRevistas)
      if ListaRevistas =/= empty .

rl [consulta] :
      listado((Codigo, empty))
      < Rev : Revista | codigo : Codigo,
                        nombre : Nombre >
      < redo : Resultado | rdo : ListaNombres >
      =>
      < Rev : Revista | >
      < redo : Resultado | rdo : sort(Nombre ListaNombres) > .

op listadoRev : -> Msg .

crl [consulta] :
      listadoRev
      < Por : Portal | listaRevistas : ListaRevistas >
      =>
      < Por : Portal | >
      < redo : Resultado | rdo : nil >
      listado(ListaRevistas)
      if ListaRevistas =/= empty .

rl [consulta] :
      listadoRev
```

```
            < Por : Portal | listaRevistas : empty >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : "No hay revistas" > .

      msg listadoMatRev : Nat Set{String} -> Msg .

      crl [consulta] :
            listadoMatRev(IMateria1, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre,
                              materia1 : IMateria1 >
            < redo : Resultado | rdo : ListaNombres >
            =>
            < Rev : Revista | >
            < redo : Resultado | rdo : sort(Nombre ListaNombres) >
            listadoMatRev(IMateria1, ListaRevistas)
            if ListaRevistas =/= empty .

      crl [consulta] :
            listadoMatRev(IMateria1, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              materia1 : IMat >
            =>
            < Rev : Revista | >
            listadoMatRev(IMateria1, ListaRevistas)
            if ListaRevistas =/= empty and IMateria1 =/= IMat .

      rl [consulta] :
            listadoMatRev(IMateria1, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre,
                              materia1 : IMateria1 >
            < redo : Resultado | rdo : ListaNombres >
            =>
            < Rev : Revista | >
            < redo : Resultado | rdo : sort(Nombre ListaNombres) > .

      crl [consulta] :
            listadoMatRev(IMateria1, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                              materia1 : IMat >
            =>
            < Rev : Revista | >
            if IMateria1 =/= IMat .

      msg listadoMat : String -> Msg .

      crl [consulta] :
            listadoMat(Materia1)
            < Por : Portal | listaRevistas : ListaRevistas,
                             listaMaterias1 : (IMateria1 |->
(Materia1, IArea), ListaMaterias1) >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : nil >
            listadoMatRev(IMateria1, ListaRevistas)
            if ListaRevistas =/= empty .

      crl [consulta] :
            listadoMat(Materia1)
```

```
            < Por : Portal | listaMaterias1 : ListaMaterias1 >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : "No existe la materia" >
            if not Materia1 in imMaterias(ListaMaterias1) .

      rl [consulta] :
            listadoMat(Materia)
            < Por : Portal | listaRevistas : empty >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : "No hay revistas" > .

      var IAr : Nat .

      msg listadoAreaRev : Nat Set{String} -> Msg .

      crl [consulta] :
            listadoAreaRev(IArea, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre,
                              materia1 : IMateria1 >
            < redo : Resultado | rdo : ListaNombres >
            < Por : Portal | listaMaterias1 : (IMateria1 |-> (Materia1,
IArea), ListaMaterias1) >
            =>
            < Por : Portal | >
            < Rev : Revista | >
            < redo : Resultado | rdo : sort(Nombre ListaNombres) >
            listadoAreaRev(IArea, ListaRevistas)
            if ListaRevistas =/= empty .

      crl [consulta] :
            listadoAreaRev(IArea, (Codigo, ListaRevistas))
            < Rev : Revista | codigo : Codigo,
                              materia1 : IMateria1 >
            < Por : Portal | listaMaterias1 : (IMateria1 |-> (Materia1,
IAr), ListaMaterias1) >
            =>
            < Por : Portal | >
            < Rev : Revista | >
            listadoAreaRev(IArea, ListaRevistas)
            if ListaRevistas =/= empty and IArea =/= IAr .

      rl [consulta] :
            listadoAreaRev(IArea, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
                              nombre : Nombre,
                              materia1 : IMateria1 >

            < redo : Resultado | rdo : ListaNombres >
            < Por : Portal | listaMaterias1 : (IMateria1 |-> (Materia1,
IArea), ListaMaterias1) >
            =>
            < Por : Portal | >
            < Rev : Revista | >
            < redo : Resultado | rdo : sort(Nombre ListaNombres) > .

      crl [consulta] :
            listadoAreaRev(IArea, (Codigo, empty))
            < Rev : Revista | codigo : Codigo,
```

```
                                        materia1 : IMateria1 >
            < Por : Portal | listaMaterias1 : (IMateria1 |-> (Materia1,
IAr), ListaMaterias1) >
            =>
            < Por : Portal | >
            < Rev : Revista | >
            if IArea =/= IAr .

      var C : Map{Nat, String} .
    op imAreas : Map{Nat, String} -> Set{String} .
    eq imAreas(empty) = empty .
    eq imAreas((I |-> A, C)) = A, imAreas(C) .

      var ListaAreas : Map{Nat, String} .
      var Area : String .

      msg listadoArea : String -> Msg .

      crl [consulta] :
            listadoArea(Area)
            < Por : Portal | listaRevistas : ListaRevistas,
                                    listaAreas : (IArea |-> Area,
ListaAreas) >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : nil >
            listadoAreaRev(IArea, ListaRevistas)
            if ListaRevistas =/= empty .

      crl [consulta] :
            listadoArea(Area)
            < Por : Portal | listaAreas : ListaAreas >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : "No existe el area" >
            if not Area in imAreas(ListaAreas) .

      rl [consulta] :
            listadoArea(Area)
            < Por : Portal | listaRevistas : empty >
            =>
            < Por : Portal | >
            < redo : Resultado | rdo : "No hay revistas" > .

      var Conf : Configuration .

      op verRevCod : String Configuration -> Configuration .
      eq verRevCod(Codigo, < rev : Revista | codigo : Codigo > Conf) =
< rev : Revista | > verRevCod(Codigo, Conf) .
      eq verRevCod(Codigo, < fas : Fasciculo | codigo : Codigo > Conf)
= < fas : Fasciculo | > verRevCod(Codigo, Conf) .
      eq verRevCod(Codigo, Conf) = none [owise].

      op verRev : String Configuration -> Configuration .
      eq verRev(Nombre, < rev : Revista | codigo : Codigo, nombre :
Nombre > Conf) = verRevCod(Codigo, < rev : Revista | > Conf) .

      op verFas : String Configuration -> Configuration .
      eq verFas(Fasc, < fas : Fasciculo | fasciculo : Fasc > Conf) = <
fas : Fasciculo | > verFas(Fasc, Conf) .
```

```
      eq verFas(Fasc, < art : Articulo | fasciculo : Fasc > Conf) = <
art : Articulo | > verFas(Fasc, Conf) .
      eq verFas(Fasc, Conf) = none [owise].

      op verRdo : Configuration -> Configuration .
      eq verRdo(< redo : Resultado | > Conf) = < redo : Resultado | >
.
      eq verRdo(Conf) = none [owise].

endom)


(view Tuple{X :: TRIV, Y :: TRIV} from TRIV to TUPLE[2]{X, Y} is
   sort Elt to Tuple{X, Y} .
endv)
```

Módulo de estrategias

Este módulo es el que indica el orden en el que deben ejecutarse las reglas de catalogación definidas en el módulo de sistema.

```
(smod PREU-STRAT is

      strat iniciar : @ Configuration .
      sd iniciar := one(actualizarEmbargos) ! ;
                        one(generarIndiceTitulos) ! ;
                        one(generarIndiceAutores) ! ;
                        one(suscribirse) ! .

      strat catalogar : @ Configuration .
      sd catalogar := one(nuevaRevista
                             orelse catalogarRevista
                             orelse actualizarEmbargos
                             orelse cargarArticulos
                             orelse catalogarArticulo
                             orelse catalogarFasciculo
                             orelse actualizarEmbargos
                             orelse enviarAlerta) ! .

      strat biblioteca : @ Configuration .
      sd biblioteca := iniciar ; catalogar ; one(consulta) ! .

endsm)
```

Módulo de prueba para la catalogación

Contiene un ejemplo de una estructura de datos inicial con artículos, fascículos y revistas y una serie de mensajes que simulan un proceso de catalogación.

```
(fmod PREU-TEST is

      pr PREU .
```

```
      op hoy : -> Nat .
      eq hoy = 20090316 .

      ops rev fas art por : -> Oid [ctor] .

      op portalTest : -> Configuration [memo] .
      eq portalTest =
            < por : Portal | listaRevistas : ("REMA"),
                                    listaCentros : ("fis" |->
"Fisica", "eis" |-> "Informatica", "mat" |-> "Matematicas"),
                                    listaAreas : (1 |-> "Ciencias", 2
|-> "Ciencias de la salud", 3 |-> "Ciencias sociales", 4 |->
"Humanidades"),
                                    listaMaterias1 : (1 |-> ("Fisica",
1), 2 |-> ("Geologia", 1), 3 |-> ("Informatica", 1), 4 |->
("Matematicas", 1), 5 |-> ("Quimica", 1)),
                                    listaMaterias2 : (1 |->
("Electronica", 1), 2 |-> ("Computacion", 3), 3 |-> ("Logica", 3), 4
|-> ("Algebra", 3), 5 |-> ("Analisis funcional", 3), 6 |->
("Ecuaciones diferenciales", 3), 7 |-> ("Topologia", 3)),
                                    listaClases : (1 |-> "Articulo", 2
|-> "Portada") >

            < rev : Revista | codigo : "REMA",
                                     issn : "11391138",
                                     nombre : "Revista Matematica
Complutense",
                                     centro : "mat",
                                     tembargo : 12,
                                     editor : "Servicio de
Publicaciones de la Universidad Complutense",
                                     departamento : "Facultad de
Ciencias Matematicas",
                                     email :
"revista_matematica@mat.ucm.es",
                                     idioma : "ingles",
                                     periodicidad : 6,
                                     historica : false,
                                     descripcion : "",
                                     materia1 : 4,
                                     listaFasciculos :
("REMA200400180001", "REMA200500180002"),
                                     listaSuscripciones :
("mat@ucm.es"),
                                     indiceTitulos : ("A New Approach
to Function Spaces on Quasi-Metric Spaces" "Entropy Solution for
Anisotropic Reaction-Diffusion-Advection Systems with L1 Data" "Sharp
Embeddings of Besov Spaces with Logarithmic Smoothness"),
                                     indiceAutores : ("BENDAHMANE,
Mostafa" "GURKA, Petr" "OPIC, Bohumir" "SAAD, Mazen" "TRIEBEL, Hans")
>

            < fas : Fasciculo | codigo : "REMA",
                                        fasciculo :
"REMA200400180001",
                                        anno : "2004",
                                        volumen : "0018",
                                        numero : "0001",
                                        fpublicacion : 0,
                                        embargo : false,
```

```
                                        finalizado : true,
                                        listaArticulos :
("REMA2004001800010007", "REMA2004001800010049") >

           < art : Articulo | codigo : "REMA",
                                      fasciculo : "REMA200400180001",
                                      fichero :
"REMA2004001800010007",
                                      paginaI : "0007",
                                      paginaF : "0048",
                                      titulo : "A New Approach to
Function Spaces on Quasi-Metric Spaces",
                                      descriptores : "quasi-metric
spaces, snowflaked transform, Besov spaces, atomic and subatomic
decompositions, entropy numbers, Riesz potentials",
                                      resumen : "",
                                      clase : 1,
                                      listaMaterias2 : (5, 7),
                                      listaAutores : ("TRIEBEL",
"Hans") >

           < art : Articulo | codigo : "REMA",
                                      fasciculo : "REMA200400180001",
                                      fichero :
"REMA2004001800010049",
                                      paginaI : "0049",
                                      paginaF : "0067",
                                      titulo : "Entropy Solution for
Anisotropic Reaction-Diffusion-Advection Systems with L1 Data",
                                      descriptores : "entropy
solutions, uniqueness, anisotropic parabolic equations.",
                                      resumen : "In this paper, we
study the question of existence and uniqueness of entropy solutions
for a system of nonlinear partial differential equations with general
anisotropic diffusivity and transport effects, supplemented with no-
flux boundary conditions, modeling the spread of an epidemic disease
through a heterogeneous habitat",
                                      clase : 1,
                                      listaMaterias2 : (6),
                                      listaAutores : ("BENDAHMANE",
"Mostafa"), ("SAAD", "Mazen") >


           < fas : Anejo | codigo : "REMA",
                                   fasciculo : "REMA200500180002",
                                   anno : "2005",
                                   volumen : "0018",
                                   numero : "0002",
                                   titulo : "Anejo extraordinario",
                                   fpublicacion : 20060427,
                                   embargo : false,
                                   finalizado : false,
                                   listaArticulos :
("REMA2005001800020081") >

           < art : Articulo | codigo : "REMA",
                                      fasciculo : "REMA200500180002",
                                      fichero :
"REMA2005001800020081",
                                      paginaI : "0081",
                                      paginaF : "0110",
```

```
                                        titulo : "Sharp Embeddings of
Besov Spaces with Logarithmic Smoothness",
                                        descriptores : "",
                                        resumen : "",
                                        clase : 1,
                                        listaMaterias2 :
(empty).Set{Nat},
                                        listaAutores : empty >

            catRev("REMA", 12, "Servicio de Publicaciones de la
Universidad Complutense", "Facultad de Ciencias Matematicas",
                    "revista_matematica@mat.ucm.es", "ingles", 12,
false,
                    "Publica trabajos originales o recapitulativos
cuidadosamente seleccionados de todas las areas de la Matematica.
Entre ellos, el articulo del 'Conferenciante Santalo' que recae en un
matematico de gran prestigio.",
                    "Matematicas", hoy)

            carArt("REMA", "2005", "0018", "0002", "0069", "0080")
            carArt("REMA", "2006", "0019", "0001", "0007", "0100")

            catFas("REMA", "REMA200400180001", "2005", "0018", "0001",
200509276, hoy)

            catArt("REMA", "REMA200500180002", "REMA2005001800020081",
"0081", "0110",
                    "Sharp Embeddings of Besov Spaces with Logarithmic
Smoothness",
                    "Besov spaces with logarithmic smoothness,
Lorentz-Zygmund spaces, sharp embeddings",
                    "We prove that smooth subvarieties of codimension
two in Grassmannians of lines of dimension at least six are rationally
numerically subcanonical. We prove the same result for smooth quadrics
of dimension at least six under some extra condition. The method is
quite easy, and only uses Serre's construction, Porteous formula and
Hodge index theorem",
                    "Articulo", ("Topologia"), (("GURKA", "Petr"),
("OPIC", "Bohumir")).Set{Tuple{String, String}})

            catArt("REMA", "REMA200500180002", "REMA2005001800020069",
"0069", "0080",
                    "Subcanonicity of Codimension Two Subvarieties",
                    "subcanonicity, codimension two, Grassmannians,
quadrics",
                    "We prove that smooth subvarieties of codimension
two in Grassmannians of lines of dimension at least six are rationally
numerically subcanonical. We prove the same result for smooth quadrics
of dimension at least six under some extra condition. The method is
quite easy, and only uses Serre's construction, Porteous formula and
Hodge index theorem",
                    "Articulo", ("Algebra"), (("ARRONDO", "Enrique")))

            newRev("FITE", "02144557", "Fisica de la Tierra", "fis")

            actEmbargos(hoy)

            indTitulos
            indAutores
```

```
            catRev("FITE", 06, "Servicio de Publicaciones de la
Universidad Complutense", "Fisica de la Tierra AA I",
                    "geofimet@fis.ucm.es", "castellano", 12, false,
                    "Recoge en sus paginas temas de Geofisica,
Geodesia, Meteorologia y Oceanografia en espanyol, portugues, frances
e ingles. Cada numero es de caracter monografico.",
                    "Fisica", hoy)

            carArt("FITE", "1990", "0000", "0002", "0000", "0000")

            catArt("FITE", "FITE199000000002", "FITE19900000000020000",
"0000", "0000",  "Cubierta", "", "", "Portada", empty, empty)

            catFas("FITE", "FITE199000000002", "1990", "0000", "0002",
0, hoy)
.

endfm)

(fmod CONFIGURATION is

  sorts Attribute AttributeSet .
  subsort Attribute < AttributeSet .
  op none : -> AttributeSet  [ctor] .
  op _`,_ : AttributeSet AttributeSet -> AttributeSet [ctor assoc comm
id: none format (m! o tm! o)] .

  sorts Oid Cid Object Msg Portal Configuration .
  subsort Object Msg Portal < Configuration .
  op <_:_|_> : Oid Cid AttributeSet -> Object [ctor object format (n
r! o g o tm! ot d)] .
  op none : -> Configuration [ctor] .
  op __ : Configuration Configuration -> Configuration [ctor config
assoc comm id: none] .
  op <> : -> Portal [ctor] .

endfm)
```

Comandos de reescritura para consulta

Es el listado de comandos que se utilizaron en las pruebas de búsqueda y acceso sobre los datos catalogados.

```
*** Listado Alfabético de Revistas
(srew portalTest listadoRev using biblioteca .)

*** Listado de Revistas por Materias
(srew portalTest listadoMat("Matematicas") using biblioteca .)

*** Listado de Revistas por Area
(srew portalTest listadoArea("Ciencias") using biblioteca .)

*** Buscar revista
(search portalTest =>! C:Configuration < rev : Revista | nombre :
Nombre:String >
               such that find(Nombre:String, "Matematica", 0) > 0 .)
(srew portalTest buscarRev("Matematica") using biblioteca .)
```

```
*** Ver revista
(search portalTest =>! C:Configuration < rev : Revista | codigo :
"REMA" > .)
(rew verRevCod("REMA", portalTest) .)
(rew verRev("Revista Matematica Complutense", portalTest) .)

*** Ver fasciculo
(search portalTest =>! C:Configuration < fas : Fasciculo | codigo :
"REMA", fasciculo : "REMA200400180001" > .)
(rew verFas("REMA200500180002", portalTest) .)

*** Consultar Indice Titulos
(search portalTest =>! C:Configuration < rev : Revista | codigo :
"REMA", indiceTitulos : IndiceTitulos:List{String<} > .)
(srew portalTest indiceTitulos("REMA") using biblioteca .)

*** Consultar Indice Autores
(search portalTest =>! C:Configuration < rev : Revista | codigo :
"REMA", indiceAutores : IndiceAutores:List{String<} > .)
(srew portalTest indiceAutores("REMA") using biblioteca .)

*** Buscar articulo
(search portalTest =>! C:Configuration < art : Articulo | codigo :
"REMA", titulo : Titulo:String >
                    such that find(Titulo:String, "Spaces", 0) > 0 .)
(srew portalTest buscarArt("SAAD") using biblioteca .)
(srew portalTest buscarArt("Spaces") using biblioteca .)

*** Suscribirse
(srew portalTest suscripcion("REMA", "fdi@ucm.es") using biblioteca
.)

*** Acceso al fichero
(srew portalTest acceder("REMA", "REMA200500180002",
"REMA2005001800020069") using biblioteca .)
```

BIBLIOGRAFÍA

- **Introducción a BPM**

Havey, M. (2005), *Essential Business Process Modeling*, O'Reilly Media, Inc. "Introduction to Business Process Modeling", capítulo 1.

Turbit, N. (2005), "Business Process Modelling Overview", The Project Perfect White Paper Collection, pp. 1-8.

Bider, I. y Khomyakov, M. (1998). "Business Process Modeling - Motivation, Requirements Implementation". Object-Oriented Technology: ECOOP'98 Workshop Reader.

- **Cómo diseñar una arquitectura BPM**

Havey, M. (2005), *Essential Business Process Modeling*, O'Reilly Media, Inc. "Prescription for a Good BPM Architecture", capítulo 2.

- **Diseño de patrones**

Havey, M. (2005), *Essential Business Process Modeling*, O'Reilly Media, Inc. "Process Desing Patterns", capítulo 4.

- **Técnicas de modelado**

Aguilar-Savén, R.S. (2004), "Business process modelling: Review and framework", International Journal of Production Economics, vol. 90, nº. 2, pp. 129-149.

Giaglis, G.M. (2001), "A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques". International Journal of Flexible Manufacturing Systems.

Havey, M. (2005), *Essential Business Process Modeling*, O'Reilly Media, Inc. "The Scenic Tour of Process Theory", capítulo 3.

- **Estándares**

Havey, M. (2005), *Essential Business Process Modeling*, O'Reilly Media, Inc. "Standards", 2ª parte.

Lonjon, A. y Architect, C. (2004), "Business process modeling and standardization", BPTrends, en http://www.bptrends.com.

Wei, W., Hongwei, D., Jin, D. y Changrui, R. A. (2006), "Comparison of Business Process Modeling Methods". En: Service Operations and Logistics, and Informatics. SOLI '06. IEEE International Conference, pp. 1136-1141.

Havey, M. (2005), "What is Business Process Modeling", ONJava.com, en http://www.onjava.com/.

Business Process Modeling Notation

Altova (2009), *Altova® UModel® 2009 User & Reference Manual*, Altova, Inc.

White, S.A. (2004), "Introduction to BPMN", IBM Corporation.

Havey, M. (2005), Essential Business Process Modeling, O'Reilly Media, Inc. "BPMI Standards: BPMN and BPML", capítulo 6.

Fortis, A. (2006), "Business Process Modeling Notation - An Overview".

Gschwind, T., Koehler, J. y Wong, J. (2008), "Applying Patterns during Business Process Modeling".

Maude

Pita, I. "Guía rápida sobre ejecución de especificaciones algebraicas en Maude bajo el entorno Eclipse para estudiantes de Estructuras de Datos".

Palomino Tajuelo, M. (2004), *Reflexión, abstracción y simulación en la lógica de reescritura,* Capítulo 2.4. Tesis doctoral de la Universidad Complutense de Madrid.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí Oliet, N., Meseguer, J. y Talcott, C. (2007), *All About Maude - A High-Performance Logical Framework*, Springer. "Object-Oriented Modules", capítulo 19.

Martí Oliet, N., Meseguer, J. y Verdejo, A. (2004), "Towards a strategy language for Maude". Fifth International Workshop on Rewriting Logic and its Applications, pages 417-441, Elsevier.

Eker, S., Martí Oliet, N., Meseguer, J. y Verdejo, A. (2006), "Deduction, strategies and rewriting". The 6th International Workshop on Strategies in Automated Deduction, pp. 3-25, Elsevier.

Martí Oliet, N., Meseguer, J. y Verdejo, A. (2008), "A rewriting semantics for Maude strategies", Theoretical Computer Science, en www.elsevier.nl/locate/entcs.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí Oliet, N., Meseguer, J. y Talcott, C. (2009), "Maude Manual (Version 2.4)", http://maude.cs.uiuc.edu/.