# Patrones de Diseño BPMN para wM Mapfre

MAPFRE

El presente documento es propiedad de MAPFRE y es exclusivamente para uso interno o de cualquiera de las Entidades del Sistema MAPFRE. Queda prohibida la *reproducción total o parcial, distribución, comunicación pública y transformación, que no podrán ser realizadas sin su autorización previa y por escrito.*

Control de cambios

|  |  |  |  |
| --- | --- | --- | --- |
| **Histórico de Versiones** | | | |
| **Versión** | **Fecha** | **Resumen de los cambios** | **Autor** |
| 1.0 | 11/02/2016 | Versión Inicial | Área Responsable BPM |
| 1.1 | 2/12/2016 | Incorporación de tareas colaborativas | Área Responsable BPM |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **Autorizaciones** | | |
|  | **Nombre** | **Fecha** |
| **Preparación** |  |  |
| **Revisión** |  |  |
| **Aprobación** |  |  |
| **Distribución** |  |  |

|  |
| --- |
| **Comentarios** |
|  |

Índice

[Patrones de Diseño BPMN para wM Mapfre 1](#_Toc468438636)

[1 Tabla de Figuras 2](#_Toc468438637)

[2 Introducción 3](#_Toc468438638)

[2.1 Objeto 3](#_Toc468438639)

[2.2 Audiencia 3](#_Toc468438640)

[3 Patrones de diseño 4](#_Toc468438641)

[3.1 Retardadores del proceso 4](#_Toc468438642)

[3.2 Patrones para Vencimientos en tareas de usuario 6](#_Toc468438643)

[3.3 Patrón de Colaboraciones (dirigidas por el proceso) 10](#_Toc468438644)

[3.4 Patrón de Colaboraciones (uso de tareas colaborativas) 16](#_Toc468438645)

[3.5 Patrones Multi-instancia de Tarea 27](#_Toc468438646)

[3.6 Patrones Cancelaciones 29](#_Toc468438647)

Tabla de Figuras

[Ilustración 1 Patrón retardador sin cancelación 5](#_Toc468438872)

[Ilustración 2 Patrón retardador con cancelación 5](#_Toc468438873)

[Ilustración 3 Patrón Básico vencimiento: Acciones en proceso con autocompletado de tarea 8](#_Toc468438874)

[Ilustración 4 Ejemplo Implementación del Patrón Básico de vencimiento 8](#_Toc468438875)

[Ilustración 5 Patrón Colaboración simple síncrona en mismo modelo. 12](#_Toc468438876)

[Ilustración 6 Patrón Colaboración simple asíncrona en mismo modelo. 12](#_Toc468438877)

[Ilustración 7 Patrón Colaboración Simple CallActivity. Modelo Principal. 13](#_Toc468438878)

[Ilustración 8 Patrón Colaboración Simple CallActivity. Modelo Colaboración. 13](#_Toc468438879)

[Ilustración 9 Patrón Colaboración Compuesta en Mismo Modelo. 14](#_Toc468438880)

[Ilustración 10 Patrón Colaboración Compuesta con CallActivity. 15](#_Toc468438881)

[Ilustración 11 Patrón Tarea Colaborativa. 18](#_Toc468438882)

[Ilustración 12 Definición Tarea Colaborativa. 19](#_Toc468438883)

[Ilustración 13 Configuración Tarea Colaborativa. 20](#_Toc468438884)

[Ilustración 14 Comentarios Compartidos en Tareas Colaborativas. 20](#_Toc468438885)

[Ilustración 15 Ejemplo Interfaz de Tarea Colaborativa Madre. 22](#_Toc468438886)

[Ilustración 16 Servicio IS de Solución Obtener Colaboraciones. 22](#_Toc468438887)

[Ilustración 17 Ejemplo Interfaz Creación de Tareas Colaborativas desde Tarea Madre. 23](#_Toc468438888)

[Ilustración 18 Servicio IS de Solución Crear Colaboraciones. 23](#_Toc468438889)

[Ilustración 19 Patrón Multi-instancia Procesos Referenciados. Detalle Modelo Padre. 25](#_Toc468438890)

[Ilustración 20 Patrón Multi-instancia Procesos Referenciados. Detalle Modelo Hijo. 25](#_Toc468438891)

[Ilustración 22 Patrón Cancelación General del proceso 34](#_Toc468438892)

[Ilustración 23 Patrón Cancelación General del proceso: Detalle estado final 34](#_Toc468438893)

## Introducción

El presente documento recoge los patrones aplicables en el modelado de procesos con la notación BPMN para su automatización con la herramienta webMethods en Mapfre, y que han sido identificados a raíz de las necesidades de implementación surgidas en los diferentes proyectos realizados.

### Objeto

Los objetivos primordiales de este documento son:

* Proporcionar a las nuevas Soluciones de automatización de procesos, la solución de diseño que se ha dado a ciertas situaciones comunes que han surgido en el modelado de procesos de negocio en wM.
* Uniformidad y estandarización en el modelado de procesos con BPMN para su implementación en wM en Mapfre.

### Audiencia

Este documento está dirigido a los equipos de desarrollo encargados del modelado, diseño e implementación de procesos con la tecnología *webMethods*.

## Patrones de diseño

Un patrón de diseño es la solución propuesta que resuelve una casuística determinada modelada en el diseño funcional del proceso (ARIS-BPMN), y que es necesario resolver en webMethods.

Hay que diferenciar entre patrón e implementación. El patrón es diseño, es decir, cómo reflejar una casuística determinada en un modelo (ARIS) y la implementación es ya la propia construcción, con todos los servicios y datos necesarios, que plasman dicha casuística en el modelo de proceso, es decir, preparada ya para desplegarla y ejecutarla en el motor (*webMethods*).

A continuación se identifican los patrones de diseño:

* Retardadores
* Vencimientos
* Colaboraciones
* Multi-instancias
* Cancelaciones

### Retardadores del proceso

#### Patrón para Retardador no cancelable

Cuando un proceso debe esperar cierto tiempo entre el fin de una tarea y el inicio de la siguiente se incluye un “retardador” de tiempo. Suele plantearse este caso cuando se tiene que esperar un plazo determinado marcado por Negocio, por ejemplo el cumplimiento de un plazo legal para enviar o emitir una notificación.

Solución de diseño:

Cuando esto sea necesario se deberá diseñar un *Gateway* de unión de dos caminos, de cuales uno venga de una tarea abstracta e incorporar un *TimeOut* en el propio join.

Patrón Básico:

El retardo se modelará de la siguiente manera:

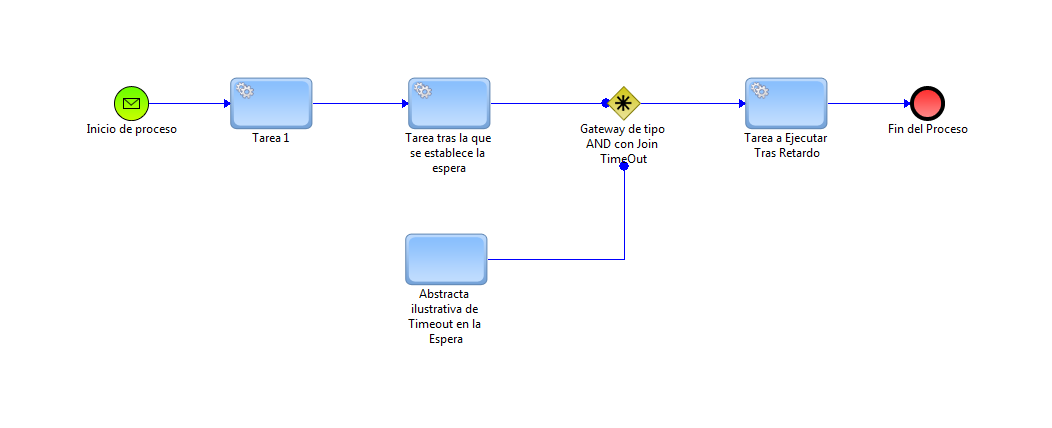


Ilustración 1 Patrón retardador sin cancelación

Inclusión de un *Gateway* de tipo *AND* tras el paso en el que se desea establecer el retardo, y al que también se accede desde una tarea abstracta por la que el proceso no pasa nunca, pero que sirve para que la condición *AND* del *Gateway* no se cumpla y sólo continúe el proceso cuando se cumpla el *Timeout* definido en dicho *Gateway*.

El tiempo de retardo debería ser calculado en el paso anterior al *Gateway* y proporcionar el plazo de espera al *Timeout*.

#### Patrón para Retardador cancelable vía Evento

Inclusión en el proceso de un “retardador” de tiempo entre dos tareas del mismo, pero se debe poder cancelar dicha espera.

Este caso se plantea cuando se establece un plazo de espera marcado por Negocio, que puede ser interrumpido si llega una confirmación externa para que se siga ejecutando el proceso.

Solución de diseño:

Se debe diseñar un *Gateway* de unión de dos caminos, con la tarea abstracta, incorporando un *TimeOut* en el propio join, y además se incorpora otra rama con un evento asociado que se ejecutará para hacer proseguir al proceso.

Patrón Básico:

Para ofrecer la posibilidad de cancelar el retardo mediante un evento del proceso, se modelará de la siguiente manera:

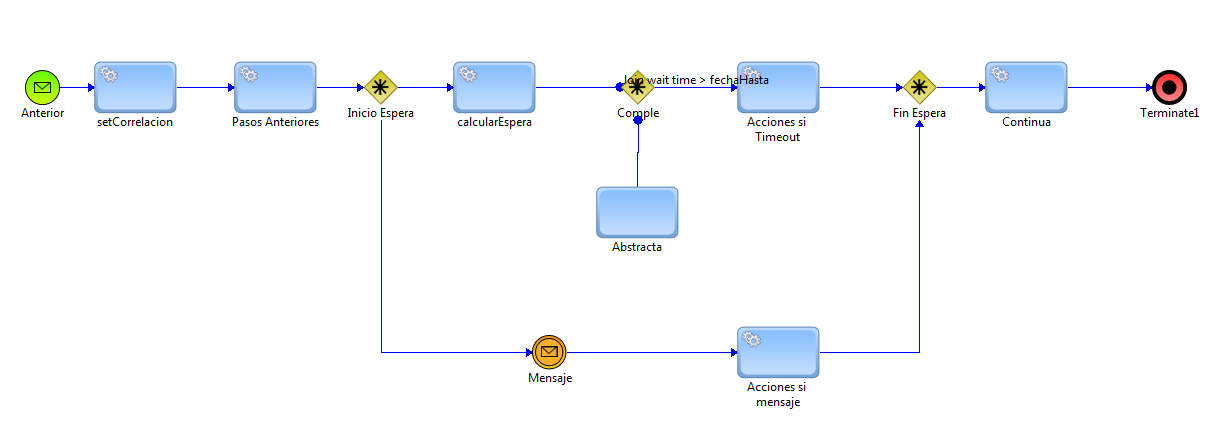


Ilustración 2 Patrón retardador con cancelación

Este caso, incluye el patrón anterior de diseño con el *Join Timeout* que fuerza el tiempo de espera, pero además se incorpora el camino del *Intermediate Message Event* utilizado para recibir la notificación de cancelación del retardo.

Si se recibe dicho evento, se continúa con el proceso, o si se cumple el tiempo de espera establecido en el *Join Timeout*. Es importante que el *Gateway* de *Inicio Espera* sea de tipo *Complex* y sobretodo que el de *Fin espera* sea de tipo *Complex* *Unsynchronized OR.*

### Patrones para Vencimientos en tareas de usuario

#### Vencimiento especifico en tareas de usuario

Se debe realizar un diseño de un vencimiento sobre las tareas de usuario, ya que la forma en la que se modela en BPMN no es trasladable directamente a wM.

Aunque la solución más similar al modelado realizado en ARIS sería incluir un Timer en la propia tarea de usuario (incluyéndola mediante un *boundary timer event* en un subproceso que contenga la tarea), esta solunión no es recomendable, ya que la “programación” asociada al timer es volátil, y se pierde la información del mismo cuando se produce cualquier reinicio en los servidores de la Plataforma.

Solución de diseño:

La solución de diseño para los vencimientos se basa en la utlización del propio evento de expiración de la tarea, cediendo el contro al proceso, en caso de que sea posible, para programar en él las acciones necesarias a realizar una vez que la tarea haya vencido.

Esta cesión de control al proceso dependerá del tipo de acciones que se desee realizar tras el vencimiento, y sobre todo de la posibilidad de que la tarea pueda ser autocompletada al expirar.

Se ofrecen dos soluciones diferentes, aplicables en función de las necesidades de cada solución:

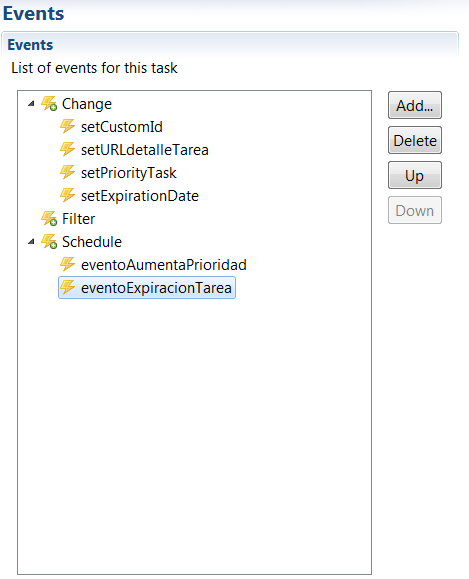
* Patrón Básico: Programar todas las acciones como tareas automáticas en el proceso, que se ejecutarían tras autocompletar la tarea una vez que expira
* Patrón Alternativo: Programar las acciones necesarias solo en el evento de expiración de la propia tarea (dependiendo del tipo de acción a realizar)

A continuación se detallan ambos patrones.

Patrón Básico: Acciones en el proceso con autocompletado de tarea

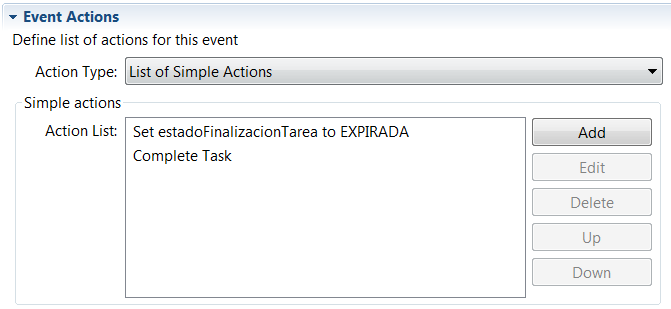
Eevento de expiración se programa en la definición de la tarea y se ejecuta en el motor de tareas.

A continuación se muestra un ejemplo concreto: al crear la tarea se calcula la fecha de expiración con la opción *“setExpirationDate”*. Cuando se cumple la duración establecida, salta el evento *“eventoExpiraciónTarea”*.



En dicho evento se programa la acción de completar la tarea, actualizando además un atributo de negocio “estadoTarea” en el que se refleja el estado de la tarea. De esta forma, en el proceso se puede determinar qué ha sucedido con la tarea manual y en función de eso se ejecutan las acciones necesarias para cada caso.

Las acciones en el evento de expiración podrían ser del tipo:



El modelado del proceso, en forma de patrón, sería similar al siguiente, para distinguir en el proceso, el motivo por el que se ha completado la tarea, y realizar acciones diferentes en él:

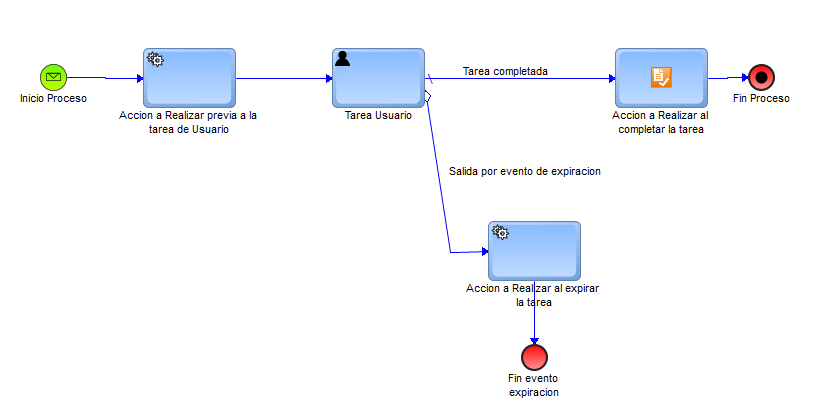


Ilustración 3 Patrón Básico vencimiento: Acciones en proceso con autocompletado de tarea

*Ejemplo:*

En el siguiente ejemplo, para distinguir el camino a seguir, se pregunta por el campo de *“EstadoTarea”* (no confundir con el propio estado de la tarea en el motor que es “*Completada*”) actualizado en el evento, tras completar la tarea Tarea\_dummy\_uno.

Según el motivo que ha producido que se complete la tarea, tras el segundo *Gateway Complex*, se pueden implementar las acciones que que se han definido por Negocio mediante la ejecución de la tarea automática “Gestionar Expiración Tarea” utilizando una rama del proceso independiente.

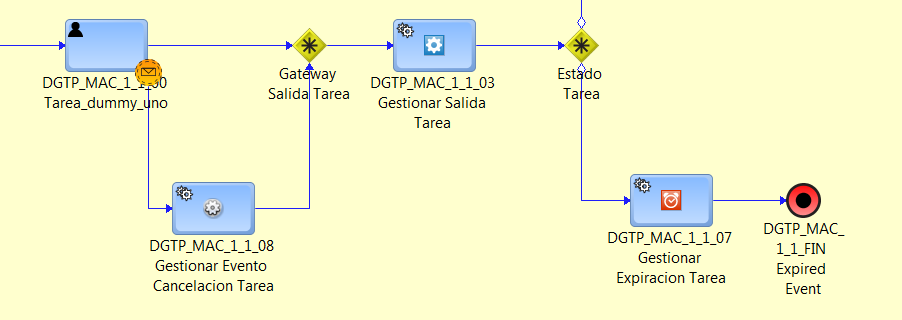


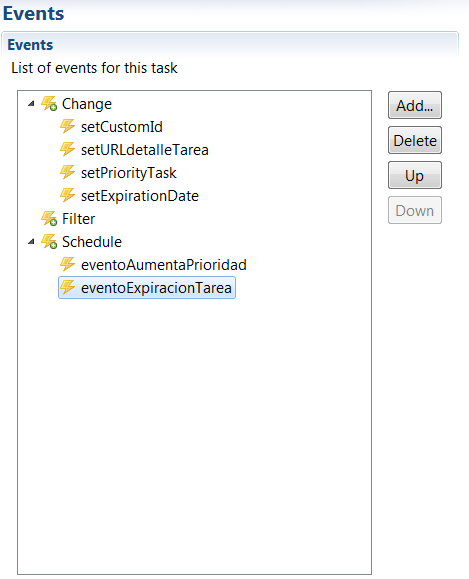
Ilustración 4 Ejemplo Implementación del Patrón Básico de vencimiento

La implementación de este patrón proporcionará la ejecución de las acciones necesarias una sola vez, cuando se cumpla el tiempo del vencimiento y se autocomplete una tarea en concreto.

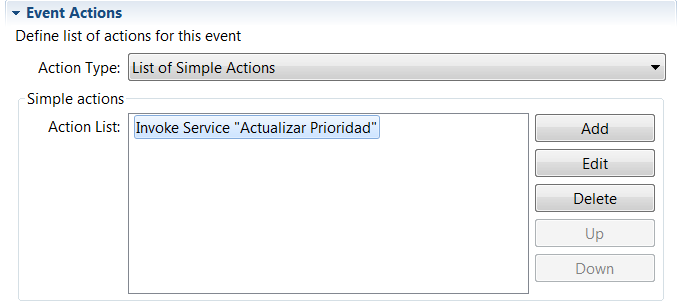
Patrón Alternativo: Acciones en tarea sin autocompletado

Para el caso en que la expiración de la tarea se desee que no se complete (es decir, se debe realizar alguna acción, pero la tarea debe permanecer pendiente para el usuario), se ofrece como alternativa la implementación de las acciones necesarias directamente en el evento de expiración de la tarea, sin pasar por completarla.

Igual que en el caso anterior, al crear la tarea se ha puede calcular la fecha de expiración con la opción *“setExpirationDate”* y cuando se cumple el tiempo establecido, salta el evento *“eventoExpiraciónTarea”* pero sin incluir el completado de la tarea e implementando solo las acciones permitidas programandolas directamente en la tarea, sin tener reflejo en el proceso.



Ejemplo: Implementar un vencimiento que haga que se produzca un aumento de prioridad pero que siga pendiente para el usuario:



Esta solución de diseño está orientada a un tipo de acciones concretas, en las cuales las acciones a ejecutar aplican a la propia tarea de usuario y a la interacción del usuario con la misma. Sirvan como ejemplo:

* Modificar datos de la propia tarea, como un cambio de prioridad.
* Notificaciones al usuario, como un envío de mail, mediante la invocación del correspondiente servicio web.

El tipo de acciones a incluir en los eventos de la tarea, debe ser contrastado con el **Área de Integración BPM** (mediante una petición de Soporte en *Service Manager*), ya que inicialmente no se recomienda incluir en los Eventos de Tarea invocaciones a servicios que soporten transacciones u operaciones con significado de negocio, recomendándose que sean modelados en el propio proceso.

Para el caso del envío de mail, deberá ser *servicio core* de la solución, y ser publicado como WS para que pueda ser invocado desde la tarea.

La implementación de este patrón facilitará la ejecución de las acciones necesarias más de una vez, si por ejemplo se vuelve a programar el tiempo de expiración de nuevo.

#### Patrón Vencimiento condicional en tareas de usuario

Se ha detectado una necesidad especifica, en algunos procesos, de incluir tareas manuales en las cuales se asocie o no un vencimieno en base a ciertos datos de Negocio. En este caso, en tiempo de ejecución se decide si la tarea de usuario se debe crear con vencimiento o sin él.

Solución de diseño:

Para esta casuística bastaría con utilizar cualquiera de los dos Patrones de vencimiento mencionados. Y cuando el dato de negocio indica que la tarea debe ser creada sin vencimiento, se creará sin la fecha de expiración rellena.

### Patrón de Colaboraciones (dirigidas por el proceso)

Cuando desde Negocio se desea que desde una tarea humana se pueda solicitar a otros usuarios que aporten cierta información para poder completarla, se puede implementar una Colaboración.

Normalmente no se tiene predefinido el número de colaboraciones en tiempo de diseño, ya que es un dato que suele determinarse en tiempo de ejecución del mismo, ya desde la propia tarea de usuario. Las colaboraciones suelen crease desde la propia interfaz de usuario de la tarea, en la que éste indica a quién se le asignará la tarea de colaboración necesaria.

Las colaboraciones asociadas a una tarea humana pueden ser de un mismo tipo de tarea, o puede ser necesario que se lancen tipos de tarea diferentes según el rol. Por otro lado, en algunos casos se requiere esperar a que los colaboradores terminen su tarea antes de finalizar la tarea origen, y otras no.

En los modelos BPMN en ARIS se suele representar este diseño con un *Conditional event* (Evento condicional) en la tarea principal, pero este diseño no es trasladable a *webMethods*:



A continuación se proponen patrones diferentes dependiendo de si la colaboración es simple, se lanza sólo una tarea humana, o es compleja, es decir, requiere más de una tarea manual y, por tanto, más lógica de proceso.

Inicialmente ambos enfoques se plantean para el caso en que todas las colaboraciones requeridas sean del mismo “tipo de colaboración”.

Por ultimo, se plantea la casuística de ambos patrones para el caso de que las tareas de las colaboraciones sean diferentes, es decir, haya varios “tipos de colaboraciones”.

En resumen, se exponen los casos:

* Patrón Colaboración simple (solo una tarea)
* Patrón Colaboración compleja (mas de una tarea)
* Patrón Colaboración simple para varios tipos de colaboraciones
* Patrón Colaboración compleja para varios tipos de colaboraciones

Tras la exposición de los patrones se detalla las diferencias que puede haber a la hora de implementar uno u otro.

#### Patrón Colaboraciones simple

Solución de diseño:

Esta solución de diseño, soporta el lanzamiento de las colaboración desde la propia interfaz de usuario. Desde ella se solicitarán las colaboraciones correspodientes y se publicará un documento para que pueda ser recogido en el mismo modelo de proceso por cada colaboración.

Lo importante de este patrón es que quede reflejado en el modelo de proceso donde se encuentra la tarea humana origen de las colaboraciones para que se pueda ver lo que courre al lanzarlas.

Este patrón está orientado a lanzar el mismo tipo de tarea para todas colaboraciones.

El hecho de que la tarea que lanza las colaboraciones pueda ser completada o no, hasta que las colaboraciones finalicen, no modifica el patrón, ya que podrá ser controlado por estado de negocio en la propia interface de usuario y además, si se necesita, en el mismo modelo proceso.

A continuación se muestran los dos formas de implementar el patrón, según haya que esperar para completar la tarea principal o no, identificándose la primera opción como “síncrona” y la segunda con “asíncrona”.

Patrón Básico Colaboración “síncrona”:

La propuesta consiste en realizar un control de las tareas a través del formulario de la tarea, en el ejemplo se puede ver como “Tarea Principal”.

En este caso concreto de colaboración “síncrona”, se plantea el caso de colaboración sin permitir que el usuario termine la tarea principal hasta que las colaboraciones de tipo A se hayan realizado.

La interfaz de la tarea debería controlar el **estado de negocio,** es decir, que se han completado todas las tareas de tipo A lanzadas. El modelo del proceso quedaría algo así:

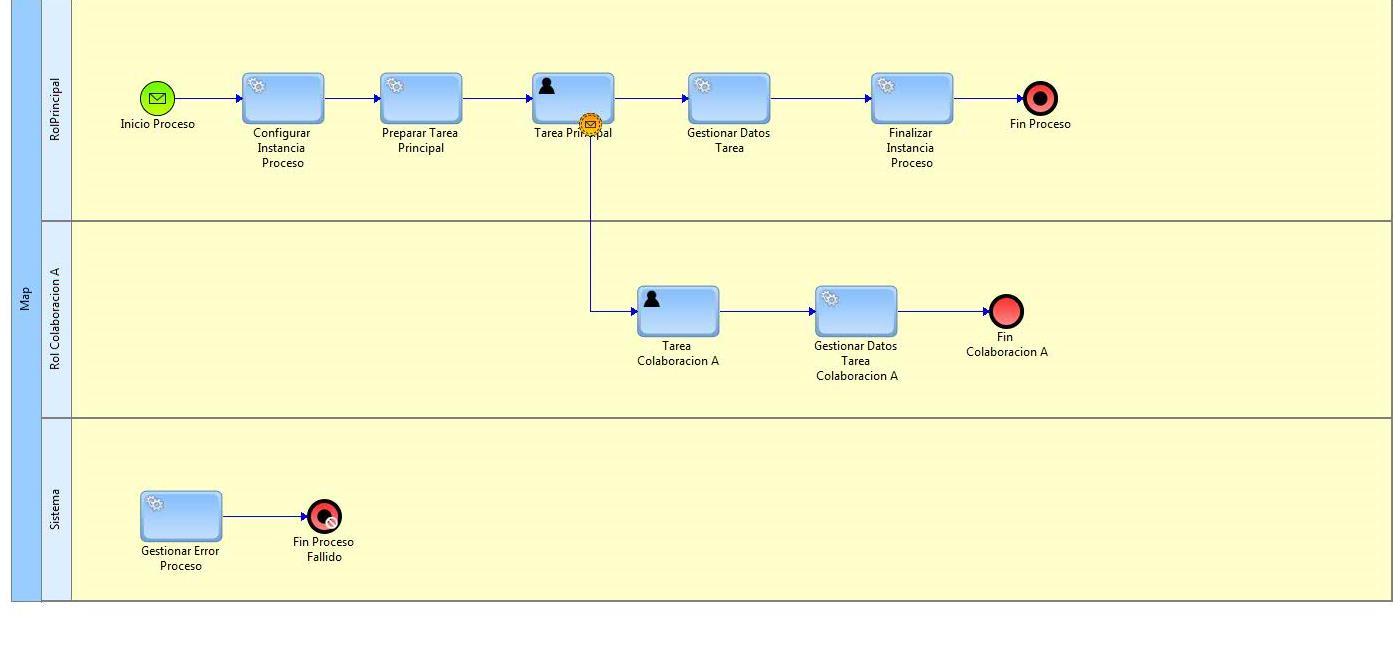


Ilustración 5 Patrón Colaboración simple síncrona en mismo modelo.

Patrón Básico Colaboración “asíncrona”:

Existe otra necesidad de modelado orientada a lanzar las colaboraciones pero pudiendo completar la tarea principal aunque no estén finalizadas las colaboraciones.

Este caso podría darse por ejemplo, si se quiere lanzar las colaboraciones mientras se trabaja y completa la tarea principal, y luego ya posteriormente, en algún otro momento del proceso, controlar la respuesta de las colaboraciones antes de proseguir. Se identifica como “asíncrona” por el hecho de no tener que esperar a las colaboraciones para completar la tarea principal.

El modelo de proceso para una sola colaboración quedaría de forma similar al siguiente modelo:

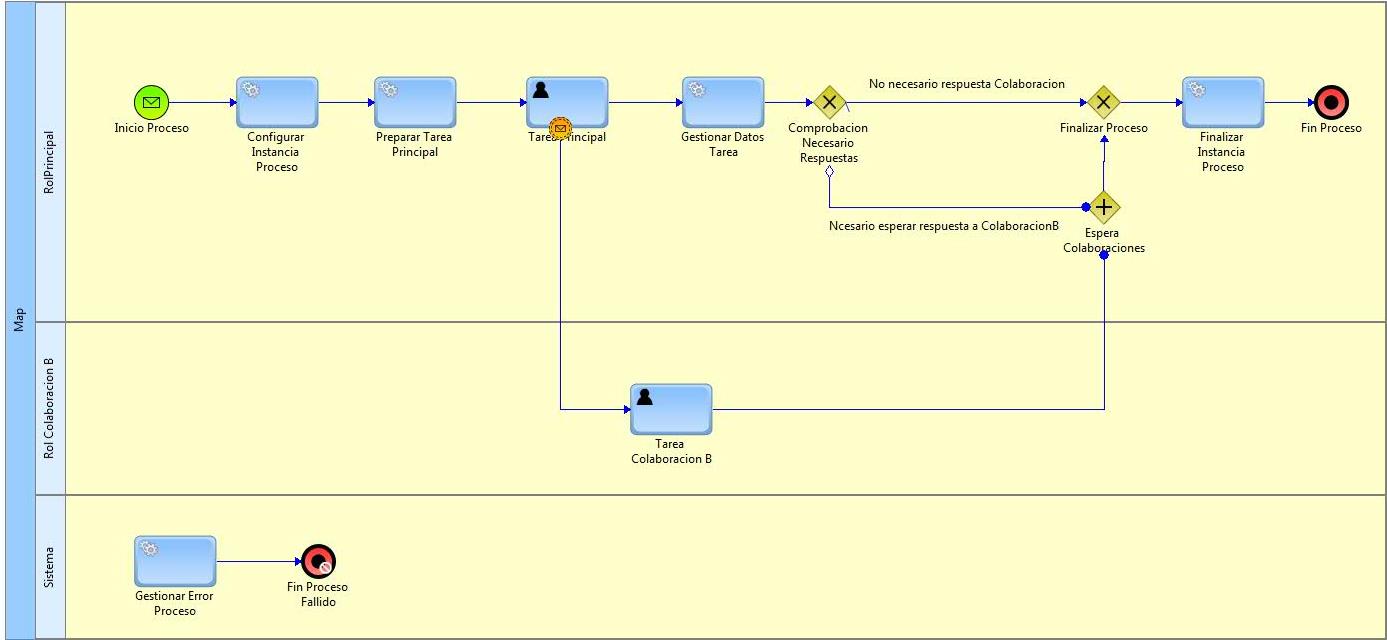


Ilustración 6 Patrón Colaboración simple asíncrona en mismo modelo.

Si se tienen varias colaboraciones, se debería trasladar al modelo de datos de negocio el número de colaboraciones solicitado y en algún punto posterior del modelo de proceso, controlar si se han realizado todas las colaboraciones. Dependerá de los requisitos del modelo de proceso, pero inicialmente parece lógico que exista tarea de usuario posterior, que revise todas las colaboraciones o respuestas, partiendo de la información que se haya grabado en propio modelo de datos de negocio al finalizar cada colaboración. Si no hay tarea de revisión, se puede controlar con la publicación de algún evento al completar la última tarea de colaboración, de tal manera, que se reciba en el punto del proceso adecuado.

#### Patrón Colaboraciones compleja

Solución de diseño:

Hay que tener en cuenta que la solución de diseño está encaminada a dar solución al modelado cuando, la colaboración sea compleja y vaya a requerir más de una tarea manual.

A continuación se muestra un ejemplo de caso aplicable para una colaboración de un mismo tipo de tarea pero con varias actividades.

Se utiliza de forma similar al patrón anterior, pero incorporando en lugar de las tareas directamente, un *Call Activity* donde ya se encuentre la lógica completa de cada colaboración.

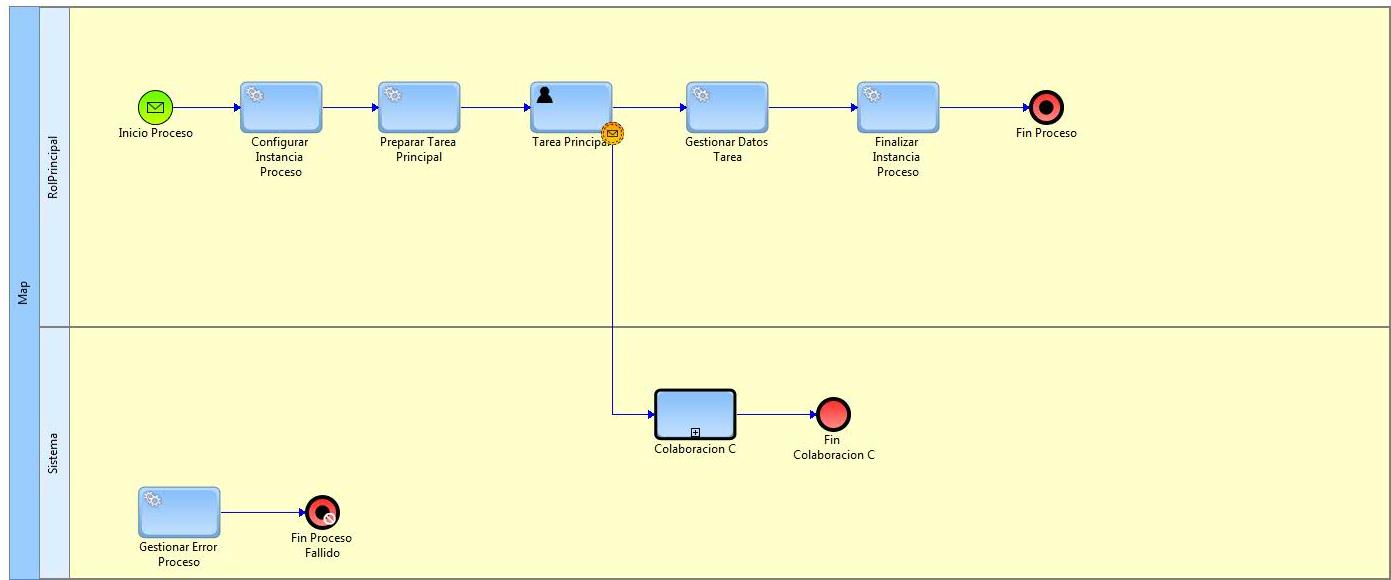


Ilustración 7 Patrón Colaboración Simple CallActivity. Modelo Principal.

El proceso a lanzar desde el *Call Activity*, para el caso de un mismo tipo de tareas:

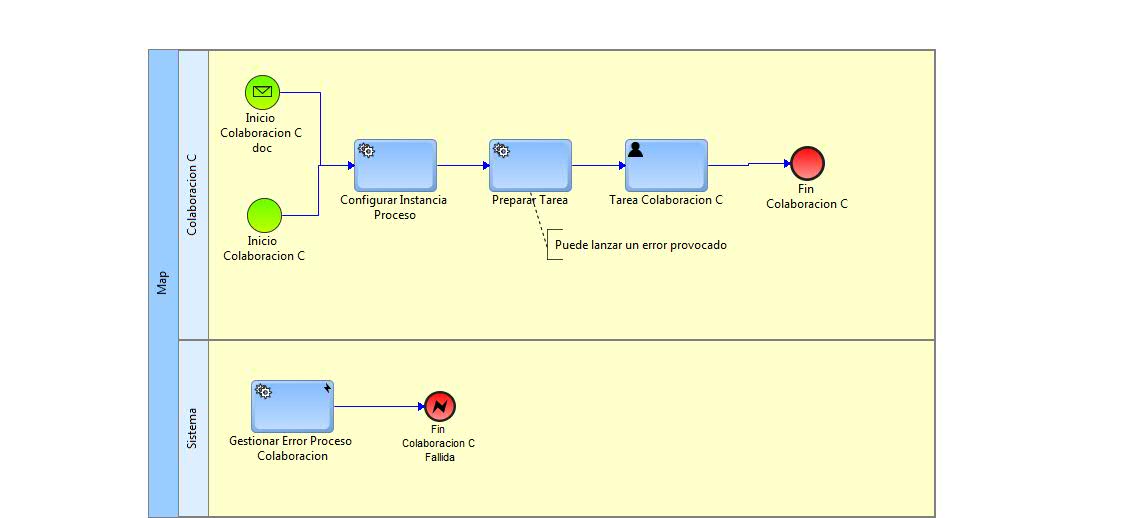


Ilustración 8 Patrón Colaboración Simple CallActivity. Modelo Colaboración.

Este modelo serviría para separar la complejidad de la colaboración en un modelo aparte que pueda tener y gestionar sus propios errores del proceso sin afectar a la tarea principal.

Para ello, es importante, que el camino del paso que captura el error del proceso finalice con un paso de tipo *End Error* (“Fin Colaboracion C Fallida”).

De esta manera, los errores se pueden controlar solo en cada colaboración.

#### Colaboración simple con distintos Tipos de Tarea

Para este caso, se ha identificado una casuística especial en el que por negocio se necesita que las colaboraciones que se requieren no son todas del mismo tipo de colaboración. Por lo tanto, se debe a tener que lanzar varios tipos de tareas. En el modelo de proceso de ARIS nos encontraríamos con varios Conditional Event (Evento condicional) en la tarea principal, cada uno conectado a su propia calle, con su propio tipo de tarea y rol asociado, para realizar las accones determinadas por cada tipo de colaboración.

Inicialmente, la propuesta es lanzar las colaboraciones de la misma forma que la colaboración simple, en el mismo modelo de proceso, desde la propia Interface de usuario, mediante la publicación de tantos documentos como sean necesarios, uno por tipo de colaboración o el mismo documento con filtro por tipo de colaboración.

Al trasladar el modelo a wM se modelara de la misma forma que las colaboraciones simples del mismo tipo de tarea pero con varias calles.

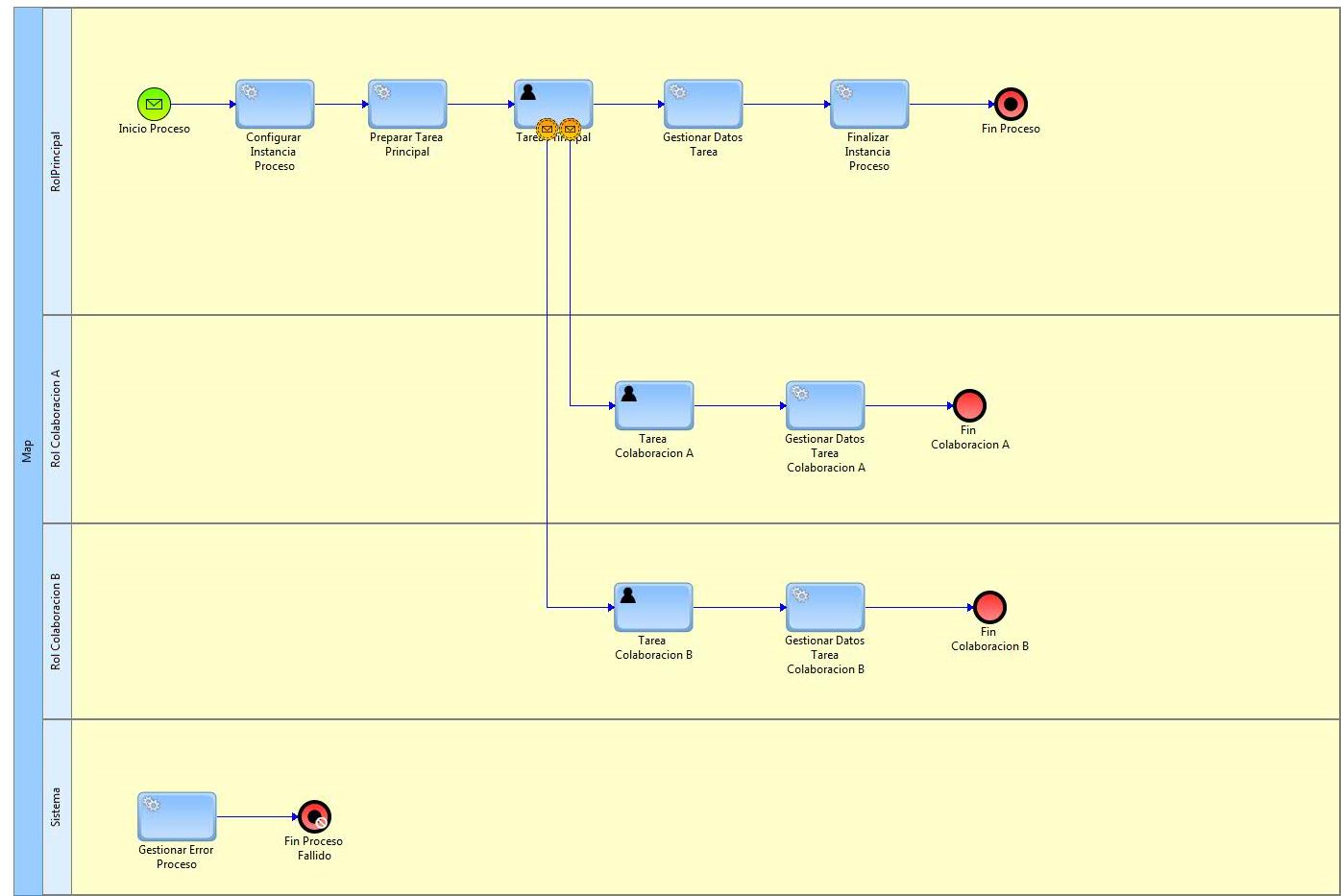


Ilustración 9 Patrón Colaboración Compuesta en Mismo Modelo.

#### Colaboración compleja con distintos Tipos de Tarea

También se podría realizar lo mismo pero incorporando cada colaboración en su propio *Call Activity*, si las colaboraciones además de diferentes, ,colaboracones complejas y tuviesen que ejecutar varias tareas, aplicando en este caso el patrón indicado de “Colaboración con *Call Activity*”.

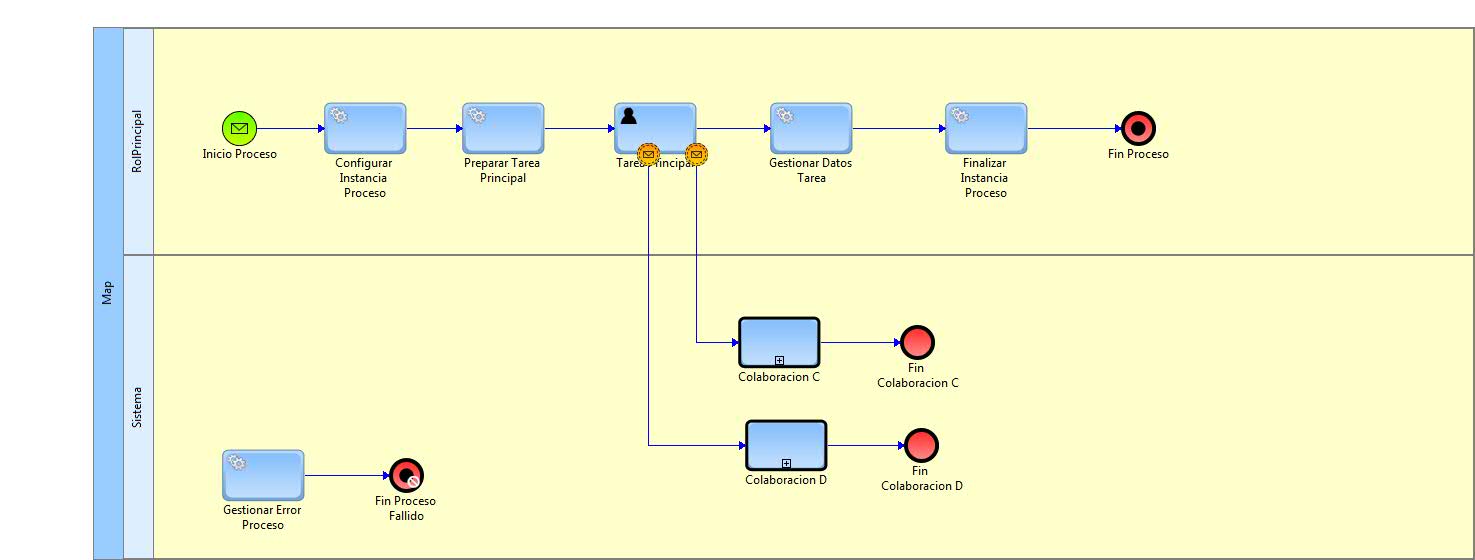


Ilustración 10 Patrón Colaboración Compuesta con CallActivity.

#### Consideraciones entre colaboración simple o compleja

De forma general, para las casuísticas identificadas, se han identificado dos patrones de diseño diferente en cuanto a modelar las colaboraciones en el mimo modelo de procesos o modelarlo en un proceso aparte e invocarlo con *Call Activity*, es decir, colaboraciones simples o complejas.

A continuación se mencionan algunas consideraciones entre un diseño u otro, que se deben tener en cuenta a la hora de decidir el diseño a realizar.

|  |  |
| --- | --- |
| Colaboración simple (tarea en el mismo modelo) | Colaboración compleja (modelado en *Call Activity* |
| Finalización de la rama o proceso principal | |
| Se finaliza el proceso como Completed.  Aunque la tarea de colaboración lanzada se muestra con un simbolito Play en el diagrama del proceso y su estado en el Resumen de Pasos es Task Queued, en realidad en el Task Engine esta completada. | Se finaliza el proceso como Completed.  El *BPMN Callable Process* no se cancela cuando su proceso padre se completa.  Se debe realizar programáticamente la cancelación del proceso de colaboración en el punto que se considere necesario, ya que existe el peligro que completen la tarea del *BPMN Callable* después y si da error el proceso principal se quede fallido. |
| Actuación ante posible fallo en la rama o proceso de colaboración | |
| El proceso principal queda fallido  Con la tarea principal activa, si se produce un fallo en cualquier tarea de la rama de colaboración (ya sea en la propia tarea manual de colaboración o en las tareas técnicas necesarias), el proceso se queda fallido haciendo que la tarea principal que estaba activa también pase a estado fallido.  Si hubiese más de una rama de colaboración también se verían afectas el resto de tareas manuales de colaboración.  Si el proceso no tuviera marcado ninguna tarea como resubmitible solo se podría resubmitir la tarea fallida que corresponde a una rama de colaboración, por lo que el único camino del proceso que avanzaría es el de la colaboración, pero la tarea principal ya no podría activarse de nuevo.  Sería aconsejable que la tarea principal o alguna tarea previa a la tarea principal fuese resubmitible para evitar estas situaciones. Aunque hay que tener en cuenta que si los datos de las colaboraciones no se han persistido en base de datos y solo estaban en el pipeline del proceso, al resubmitir la tarea se habrían perdido y se volvería a los datos de inicio de creación de la tarea principal. | El proceso principal sigue activo.  El proceso colaboración es el que está fallido.  La tarea principal no se ve involucrada en el fallo de la colaboración y podría completarse sin problemas finalizando el proceso principal. Incluso en el paso de Gestionar Error Proceso Colaboración se podría informar a la Tarea Principal del error ocurrido en la colaboración solicitada, para ser tratado según se considere.  La solución es sencilla, resubmitir el paso fallido del proceso de colaboración y todo sigue activo |
| Características propias de la implementación | |
| Es aconsejable que la tarea principal se marque como resubmitible | Basta con crear la tarea colaboración en el proceso como resubmitible |
|  | Importante que el camino del proceso correspondiente al paso que captura el error del proceso finalice con un paso de tipo End Error para no escalar el fallo del proceso al proceso padre. |

El hecho de elegir entre un modelado u otro, dependerá de la complejidad de las colaboraciones, y de donde se consideren mejor, según casuísticas concretas, la forma de controlar los errores.

### Patrón de Colaboraciones (uso de tareas colaborativas)

Como alternativa al enfoque anterior de implementación de colaboraciones desde el propio modelo de proceso existe la posibilidad de uso de la funcionalidad proporcionada por las “tareas colaborativas” de *webMethods*.

Además de las tareas de intervención humana modelizadas explícitamente en el proceso siguiendo un determinado orden de aparición, a veces existen hechos no planificados o interacciones no definidas de antemano que son requeridas para completar una determinada tarea o proceso. Para ello, los propietarios de las tareas que tengan autorización, pueden crear nuevas “tareas hijas” al vuelo, con objeto de colaborar con otros usuarios para completar la tarea que le fue asignada.

Este tipo de tareas que se crean al vuelo como “hijas” de una tarea principal o “madre” se denominan en *webMethods* como “*Collaboration Tasks*” (en adelante Tareas colaborativas).

A través de estas tareas de colaboración, el usuario que trabaja con la tarea “madre” puede solicitar la colaboración de otros usuarios, mediante la asignación de tareas hijas, de manera que finalmente, y como resultado de esa colaboración, pueda completar esa tarea principal.

Es de gran valor en este tipo de tareas colaborativas la **capacidad para compartir información con la tarea principal**, de tal modo que los datos de negocio, los comentarios e incluso documentos, se podrían compartir entre ellas.

La gestión documental, en cualquier caso, no se implementa en MAPFRE de esta manera, ya que toda lógica de gestión y de compartición documental se implementa con *Documentum*.

Sin embargo, la capacidad de estas tareas para poder compartir con la tarea principal o tarea madre los datos de negocio o los comentarios las hacen diferentes al resto de tareas de intervención humana en los procesos de negocio.

#### Características especiales de las tareas colaborativas

Las tareas colaborativas sólo difieren del resto de tareas en determinados detalles:

* La forma en la que se inician:
  + Se encolan programáticamente
    - En consecuencia, no están asociadas a un modelo de proceso en concreto.
    - Se completan automáticamente si se completa la tarea madre.
* Su relación única con la tarea madre desde la que fueron creadas:
  + Tienen informadas las propiedades relativas a la propia colaboración (Id de Colaboración, *Parent Task Id*)
* Su operación con la madre mediante un “proceso” único de colaboración:
  + Las tareas se crean directamente en el *Task Engine*. El proceso de negocio, y el motor que lo automatiza, no son conscientes de la existencia de las mismas. Sin embargo, cada uno de los procesos de colaboración derivados de la creación de tareas hijas desde una tarea principal se gestiona de forma independiente desde *MWS*.

#### Implementación de tareas colaborativas. Aspectos a considerar

Para garantizar el éxito del funcionamiento de las tareas colaborativas en MAPFRE será necesario realizar los pasos expuestos en este apartado.

##### Inclusión de tareas colaborativas en modelo de proceso como tareas abstractas

Con el fin de documentar la existencia, la posibilidad operativa de colaboración, y proporcionar la mayor información posible, se deben de representar las tareas colaborativas a nivel de modelo de proceso, en los pasos en los que pueda existir dicha colaboración.

Para ello, y siempre que se cuente con la información necesaria en tiempo de diseño, se implementará un paso de proceso de tipo Abstracto, con el nombre de la tarea colaborativa, posicionado en el *swimlane* correspondiente a la posible asignación de la tarea colaborativa, y con una transición desde la tarea madre, donde se invocarán las colaboraciones, hasta la tarea colaborativa propiamente dicha:

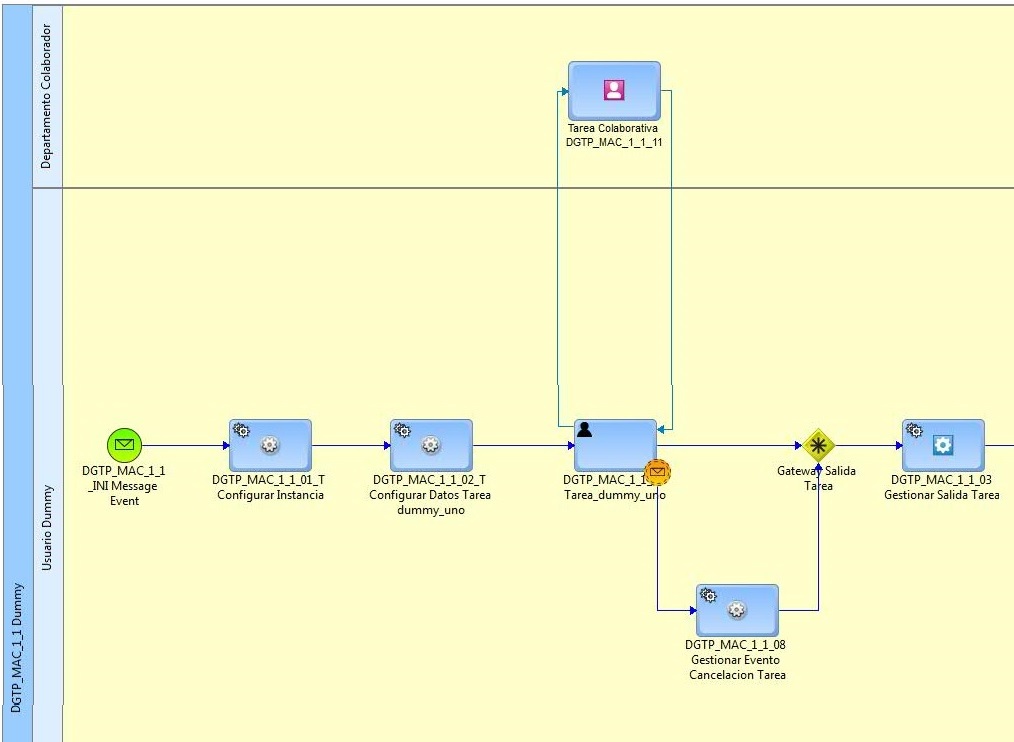


Ilustración 11 Patrón Tarea Colaborativa.

De esta manera se garantiza que el modelo de proceso proporciona visualmente la información relativa a las tareas colaborativas existentes.

Se sugiere personalizar el paso de la tarea colaborativa con una imagen de paso representativa, y cambiar de color las transiciones para diferenciarlas del resto. La transición de vuelta al paso original es optativa y no debería de representarse en el caso de que comprometa o empeore la legibilidad del modelo de proceso.

El paso de tipo Abstracto carece de implementación y no tiene efecto en tiempo de ejecución para el modelo de proceso.

##### Creación/definición de tipos de tareas colaborativas hijas, y relación con el tipo de tarea madre

En la definición de tareas colaborativas se deben de tener en cuenta los siguientes puntos.

###### Especificación del tipo de tarea madre y atributos principales de las tareas colaborativas

La implementación de un tipo de tarea colaborativa con Designer es análoga a la implementación de cualquier otra tarea de un modelo de proceso, con la particularidad de que, al menos, se debe de especificar un Tipo de Tarea Madre o *Parent Task Type*.

Es posible crear el tipo de tarea colaborativa en el mismo proyecto que el tipo de tarea madre o en otro distinto, si bien el tipo de tarea madre debe de existir en el *workspace* de *Designer* para poder ser elegida en el desplegable:

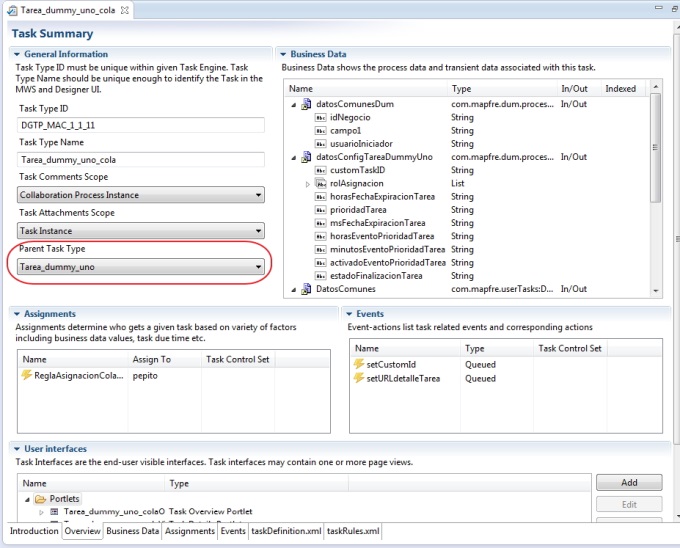


Ilustración 12 Definición Tarea Colaborativa.

La nomenclatura para el ID de tipo de tarea colaborativa (*Task Type ID*), sigue la misma norma que para las tareas normales, y normalmente vendrá heredado del diseño realizado en ARIS. El mismo concepto es aplicable al nombre de tipo de tarea (*Task Type Name*).

En cuanto a las secciones de Datos de Negocio, Asignaciones y Eventos, se pueden implementar según necesidades, siempre teniendo en cuenta las normas y eventos obligatorios de arquitectura que aplican a la implementación de tareas de procesos. Para más información consultar la documentación “*Diseño componente tareas humanas*”.

###### Especificación de datos de negocio en tareas colaborativas

Existen dos modos de funcionamiento que proporciona el sistema para los datos de negocio de tareas colaborativas:

* **Datos de negocio compartidos**: Es el comportamiento por defecto. Al crear la tarea colaborativa el sistema automáticamente saca una copia del contenido del TaskData de la tarea madre y se lo asigna como entrada a la tarea colaborativa. La tarea colaborativa puede tener otros campos de entrada distintos de los que tenga definidos la tarea madre, pero con este modo de funcionamiento siempre estarán a blanco tras el encolado. Los campos de salida de la tarea colaborativa pueden ser cualesquiera.
* **Datos de negocio no compartidos**: En este modo, es responsabilidad del desarrollador el asignar el *TaskData* de entrada para la creación de la tarea colaborativa. Tanto los campos de entrada como los de salida pueden ser iguales o distintos de los definidos en la tarea madre. Para activar este modo es necesario acceder a la consola de *My webMethods Server*, en la sección: Administración -> Negocio -> Tareas -> Administración de *Task Engine*, y desmarcar, para el Tipo de Tarea colaborativa, el check de ‘Uso compartido de datos de tareas habilitado (para colaboración)’, y hacer click en ‘Actualizar’:

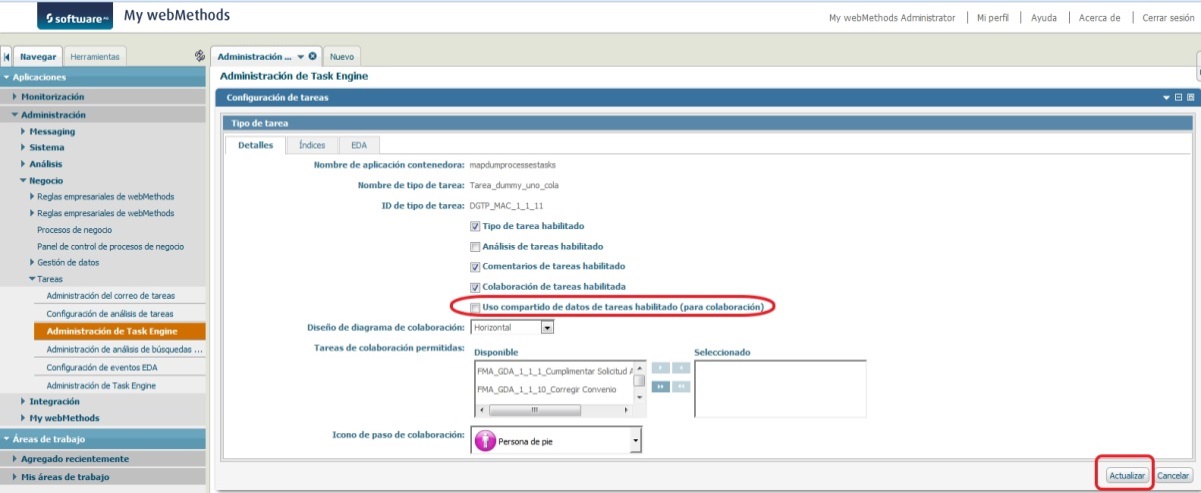


Ilustración 13 Configuración Tarea Colaborativa.

En cualquiera de los dos modos de funcionamiento, una vez instanciada la tarea colaborativa, los valores contenidos en su *TaskData* son totalmente independientes de los datos de su tarea madre.

###### Uso de comentarios compartidos en tareas colaborativas

Para las tareas colaborativas, puede ser interesante poder registrar comentarios que sean compartidos con la tarea madre.

Para activar el uso de comentarios compartidos, es necesario especificarlo mediante *Designer*, en la definición del Tipo de Tarea colaborativa:

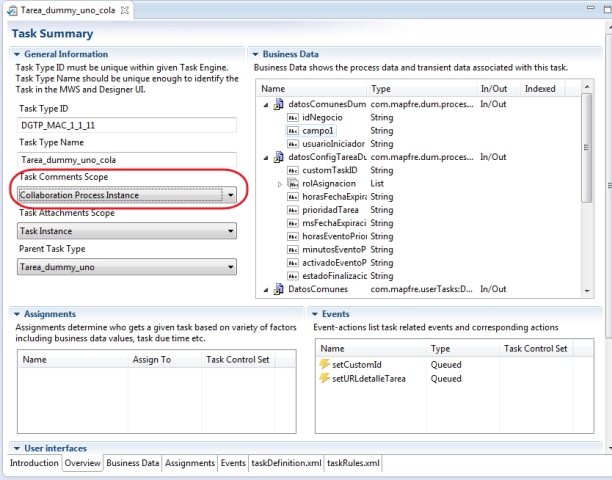


Ilustración 14 Comentarios Compartidos en Tareas Colaborativas.

De manera análoga, si se desea que los comentarios que sean creados en la tarea madre sean compartidos por sus tareas colaborativas, se deberá de fijar el mismo ámbito para los comentarios en la definición del Tipo de Tarea madre.

Para la gestión de los comentarios de tareas están disponibles los servicios *IS* de arquitectura:

* com.mapfre.bpm.api:addTaskComment
* com.mapfre.bpm.api:deleteTaskComment
* com.mapfre.bpm.api:getTaskComments
* com.mapfre.bpm.api:updateTaskComment

Que deberán de ser encapsulados en los servicios *IS* de aplicativo correspondientes a las operaciones de obtener tarea, completar tarea y/o actualizar tarea, según corresponda.

##### Inclusión de relación de tipos de tareas colaborativas en XML de configuración de la solución

Al igual que para el resto de tareas, es necesario configurar el XML de solución *(“<ID\_Solución>\_CARGA\_BTW.xml*”) para que especifique el tipo de tarea colaborativa, y el modelo de proceso al que pertenece, con el fin de que aparezca en los resultados de búsqueda de la Bandeja de Tareas Corporativa.

Por ejemplo:

<procesoTecnico> <processModelID>MapDumProcesses/DGTP\_MAC\_1\_1\_Dummy</processModelID> <nombreProceso>Dummy</nombreProceso> <Tareas> <Tarea> <taskTypeID>DGTP\_MAC\_1\_1\_11</taskTypeID> <nombreTarea>Tarea\_colaborativa\_dummy\_uno</nombreTarea> <gruposSupervisores>MapDumSupervisor</gruposSupervisores> <modoApertura>1</modoApertura> <completableBTC>N</completableBTC> <urlConsultaTarea/> <urlGestionTarea/> <rolesAutorizadosTarea> <rol>MapDumUsers</rol> </rolesAutorizadosTarea> </Tarea> … </Tareas>

</procesoTecnico>

Asimismo, es necesario completar el fichero .properties (“*<ID\_Solución>\_ApplicationResources.properties*”) correspondiente a la aplicación, con las entradas de los literales necesarios para el Escritorio de Procesos:

# Nombres de Tarea DGTP\_MAC\_1\_1\_00.name=Tarea Dummy Uno DGTP\_MAC\_1\_1\_11.name=Tarea Colaborativa Dummy Uno DGTP\_MAC\_1\_2\_03.name=Tarea Dummy Dos # Códigos de Tarea DGTP\_MAC\_1\_1\_00.label=DGTP\_MAC\_1\_1\_00 DGTP\_MAC\_1\_1\_11.label=DGTP\_MAC\_1\_1\_11 DGTP\_MAC\_1\_2\_03.label=DGTP\_MAC\_1\_2\_03

##### Implementación de operativa con las tareas colaborativas desde interfaz de usuario de las tareas madre

La gestión de las tareas colaborativas y su interacción con la tarea madre dependen por completo de su implementación en particular. La única acción automática realizada por el sistema es el auto-completado de las tareas colaborativas en el caso de completar la tarea madre. Por lo tanto, se debe de desarrollar la lógica necesaria para la interacción tanto a nivel de presentación como a nivel de servicios IS.

* Presentación de colaboraciones existentes:
  + Desde la interfaz de usuario de la tarea principal o madre se deberían poder ver las tareas hijas creadas, así como su estado, comentarios compartidos, etc., de modo que el usuario que tiene asignada dicha tarea principal pueda actuar en consecuencia, por ejemplo:

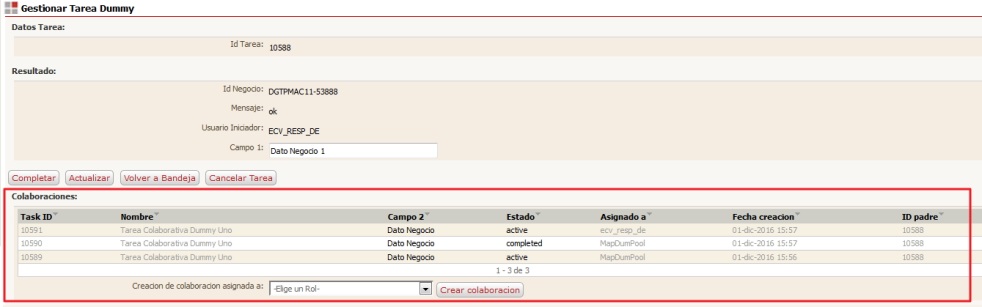


Ilustración 15 Ejemplo Interfaz de Tarea Colaborativa Madre.

* + Se deberá crear un servicio auxiliar de la tarea en el paquete *IS* *Map<Codigo\_solucion>UserTask* de la solución, en la carpeta que corresponda a la tarea madre, que pueda ser consumido vía *WS* desde la interfaz de usuario. Este servicio debería ser nombrado *obtenerColaboraciones<Nombre\_Tarea\_Madre>*. Desde este servicio se invocará al servicio del Framework *com.mapfre.bpm.api:searchTasksCollaboration*, proporcionando como entrada el ID de proceso colaborativo que aparece en el TaskInfo de la tarea madre:

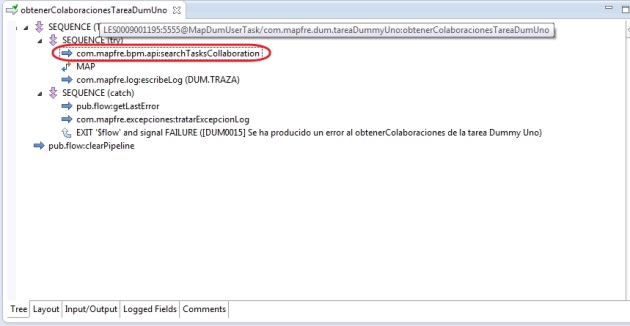


Ilustración 16 Servicio IS de Solución Obtener Colaboraciones.

* Creación de nuevas colaboraciones:
  + También será la interfaz de usuario de la tarea principal la encargada de permitir la creación de nuevas tareas colaborativas(hijas), así como su asignación, si fuera necesario:

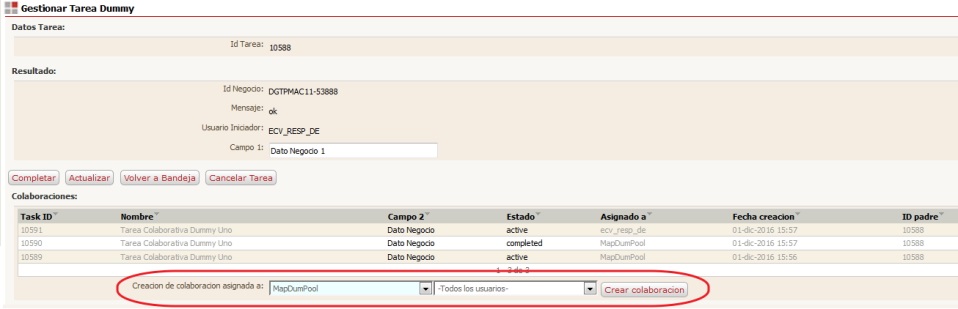


Ilustración 17 Ejemplo Interfaz Creación de Tareas Colaborativas desde Tarea Madre.

* + Se deberá crear un servicio auxiliar de la tarea en el paquete *Map<Codigo\_solucion>UserTas*k de la solución, en la carpeta que corresponda a la tarea madre, que pueda ser consumido vía WS desde la interfaz de usuario. Este servicio debería ser nombrado crearColaboracion<Nombre\_Tarea\_Colaborativa>. Desde este servicio se invocará al servicio del *Framework* *com.mapfre.bpm.api:createCollaborationTask*, proporcionando como entrada los datos de negocio necesarios para la creación de la tarea colaborativa (*TaskData*), en caso de que no estén habilitados los datos de negocio compartidos con la tarea madre, así como los datos intrínsecos de la tarea que sean necesarios (*TaskInfo*):

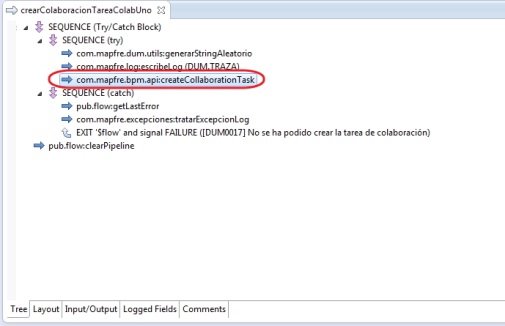


Ilustración 18 Servicio IS de Solución Crear Colaboraciones.

Hay que tener en cuenta que, si la tarea colaborativa no tiene una regla de asignación, se debe de proporcionar un valor para el campo de asignación *AssignedTo* del *TaskInfo*, que deberá de escoger el usuario a través de la interfaz de la tarea madre.

Asimismo, caso de existir más de un tipo de tarea colaborativa para el mismo tipo de tarea madre, la interfaz de usuario debería de proporcionar un contro tipo *dropdown* o similar, que permita al usuario escoger el tipo de tarea colaborativa que se va a crear.

##### Aspectos de seguridad y permisos de las tareas colaborativas

Con el fin de poder operar sobre las tareas colaborativas, es necesario asignar los permisos correspondientes a estos tipos de tarea colaborativas, de la misma forma en que se crean los permisos para cualquier otro Tipo de Tarea correspondiente a un modelo de proceso, teniendo en cuenta los posibles roles, grupos o usuarios de asignación para las mismas.

##### Posibilidades de sincronización de las tareas colaborativas

Dado que el comportamiento de las tareas colaborativas depende casi por completo de la implementación que se desarrolle, se deben gestionar los escenarios de manera específica, de forma programática, por parte de las soluciones. La única excepción es el auto-completado de las tareas hijas cuando se completa la madre.

Como escenarios genéricos de sincronización se contempla:

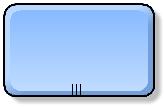
* Completado automático de hijas cuando se completa la tarea madre (comportamiento por defecto del sistema): Cuando el estado de la tarea madre pasa a ‘Completed’, todas las instancias de tareas colaborativas asociadas se marcan también como completadas, y ya no es posible realizar ninguna acción sobre ellas.
* Completado automático de la madre cuando se completan todas las hijas: Si se desea este comportamiento será necesario implementar, en el servicio de solución de *completarTareaColaborativa*, una llamada al servicio de solución *obtenerColaboraciones<Nombre\_Tarea\_Madre>* y, tras comprobar si todas las tareas colaborativas tienen estado completado, invocar al servicio de solución correspondiente de completarTarea para la instancia de tarea madre.

##### Portlets de presentación de tareas colaborativas

Es necesario crear los portlets de presentación, análogamente a como se realiza para cualquier otro tipo de tarea de un modelo de proceso. Para más información consultar el documento “*Caracteristicas Portlets CAF webMethods*” y “*Recomendaciones de diseño de componentes CAF*”.

##### Patrones Multi-instancia de Procesos

Se debe implementar una Multi-instancia cuando una misma tarea debe generarse N-veces (en paralelo o en secuencia). En ARIS-BPMN se modela como un tipo *Call Activity* (Interfaz de proceso) con el símbolo de multi-instancia:



#### Patrón Propuesto Procesos Referenciados

Solución de diseño:

La solución de diseño para modelar la multi-instancia de procesos es la utilización de procesos referenciados, es decir, *Dynamic Referenced Process* (opción de *webMethods Referenced Process*), que mediante el uso de un *Call Activity* instancia los n procesos requeridos, en lugar de utilizar la publicación/subscripción manual.

El uso de *Dynamic Referenced Process*:

* Permite invocar a múltiples procesos desde un *Call Activity*
* Puede ser síncrono o asíncrono

Detalle Patrón Procesos Referenciados

Estos procesos se lanzan desde un *Call Activity* y tratan unos documentos en concreto, en lugar de tener que programar a mano tu propio control de publicación/subscripción, como se mostraba en el patrón anterior.

Para entenderlo un poco mejor, si tenemos un Proceso Padre (“TestMultiInstanciaWM”):

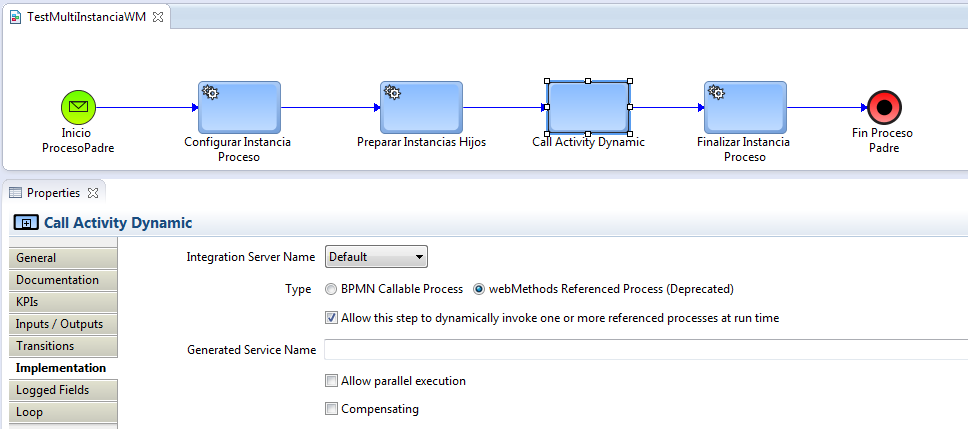


Ilustración 19 Patrón Multi-instancia Procesos Referenciados. Detalle Modelo Padre.

Y un proceso hijo A:

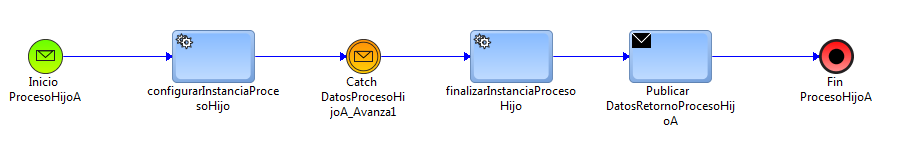
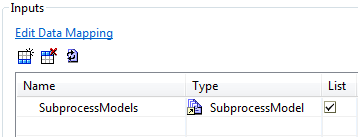


Ilustración 20 Patrón Multi-instancia Procesos Referenciados. Detalle Modelo Hijo.

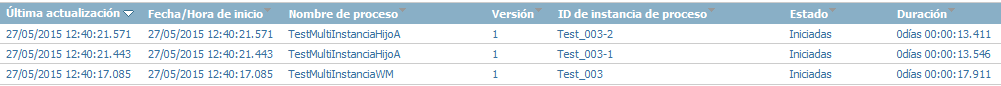
La entrada del paso del *Call Activity* de tipo *Dynamic Referenced Process* es el documento pub.prt:SubprocessModel, que en tiempo de ejecución tendrá que ser informado con los modelos de subprocesos, el número de instancias, y los documentos de entrada y salida necesarios.



Se pueden invocar a varios procesos y se puede informar, si se desea que las invocaciones sean síncronas o asíncronas por cada proceso al que se invoca.

Si se requiere respuesta del proceso hijo, el documento que se necesita de respuesta se debe enviar devuelta al padre implementado con Send Task Activities**.**

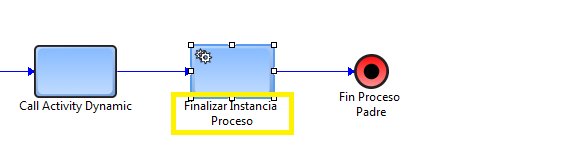
Si se lanzan por ejemplo dos instancias del hijo A:

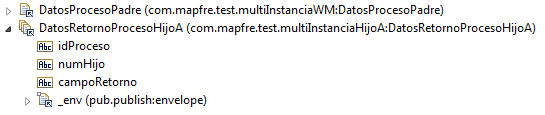


El padre recibirá tantos documentos como hijos se hayan creado y hayan terminado. La forma de poder recibirlos todos es que en tiempo de ejecución los va añadiendo a un documentList del tipo de documento que se retorna el hijo.

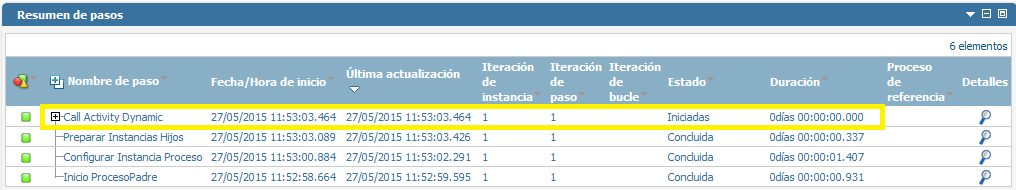
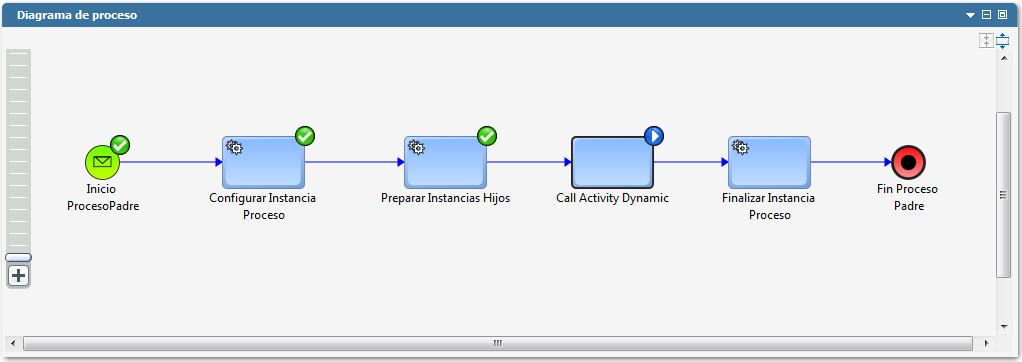


Así que en el siguiente paso después del *Call Activity* (“*Finalizar Instancia Proceso”*) si se quiere utilizar la información de retorno se tendría que declarar como un Document Referenced List.





En el seguimiento de las instancias de procesos hijos lanzadas, desde el proceso padre se podrá ver la tarea en curso desde la que se lanzan y un enlace a los hijos:





#### Consideraciones a tener en cuenta en la aplicación del patrón

A continuación se describen algunas consideraciones que se han contrastado a la hora de diseñar el patrón de procesos referenciados y que pueden servir de orientación a la hora de implementarlo.

|  |
| --- |
| **Patrón Procesos Referenciados** |
| Seguimiento de instancias de procesos en consola |
| En el **Step Summary** de ***Dynamic wM Referenced Process*** no se ve cuales se han ido completando porque se actualizan a Completed cuando finalizan todos, pero se tiene un enlace directo a la ejecución de cada uno de ellos |
| Gestión de errores: escalado de errores al padre |
| Con ***Dynamic wM Referenced Process*** existe la posibilidad de escalar el fallo de los hijos al padre de manera que si falla un hijo falle el padre, simplemente marcando el check “Escalate failed process status to parent” del nodo fin con estado “Failed” del proceso hijo. |
| Datos entre Padre-hijos |
| Con ***Dynamic wM Referenced Process*** se pueden tratar cada documento respuesta de los hijos, que estarán recogidos en el documento pub.prt:SubprocessModel que se haya definido. Pero siempre, una vez finalicen todos ellos |

### Patrones Multi-instancia de Tarea

Se ha detectado la necesidad de reflejar en webMethods la multi-instancia de tarea paralela, que puede modelarse en los modelos ARIS como elemento de tipo tarea con multiinstancia. Sería similar al siguiente:



Se ha estudiado dicha posibilidad de cara a procesos que requieran en un momento dado, la realización de una misma tarea por parte de varios Intervinientes de forma paralela, pero no lanzada, como el caso de las colaboraciones, desde la interface de usuario a demanda, sino en el propio modelo de proceso, según los casos particulares del modelo de negocio.

En estos casos los participantes o roles se suelen obtener, según los datos de negocio del proceso. Por ejemplo, lanzar una misma tarea para cada país, ramo etc…, es información que se conoce en el contexto del proceso.

Solución de diseño:

Se proponen dos patrones diferentes, utilizar los procesos referenciados expuestos en la multi-instancia de proceso para ejecutar tantas tareas(procesos) como se requieran, o simplemente modelar la casuística en el propio modelo de proceso mediante el control de bucles.

Patrón básico:

**Patrón Procesos Referenciados (ya expuesto en la multiinstancia de procesos)**

La solución de diseño serían lanzar los ‘n’ procesos refenciados, uno por cada instancia de tarea que se necesita e utilizar el control propio que ofrece el *Call Activity.*

Se debe tener en cuenta que lo normal es que el proceso padre que lanza las tareas multi-instancia, necesitará, una vez realizadas, la información obtenida en la resolución de la tareas, por lo que utilizar el documento que ya se ha visto en el patrón de procesos referenciados.

Patrón alternativo:

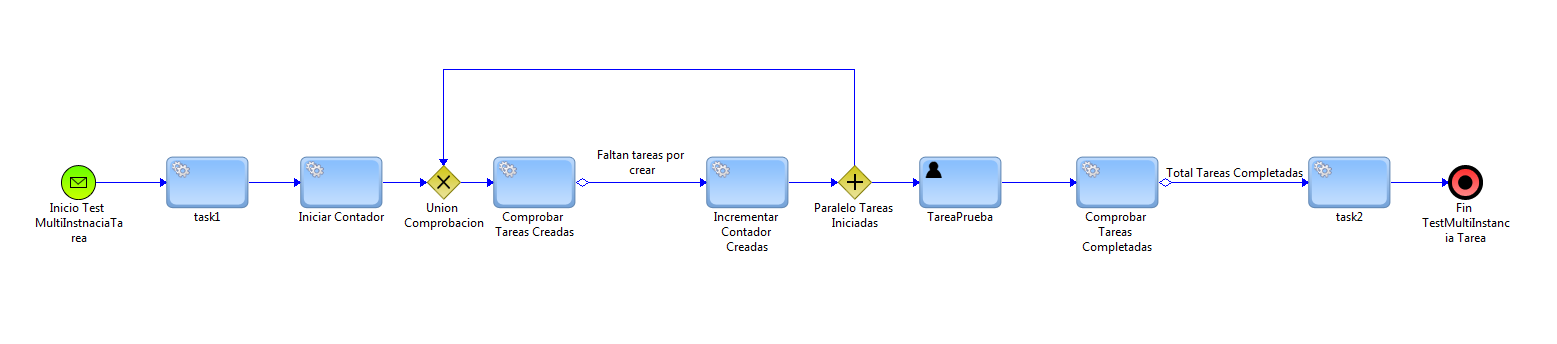
**Patrón Programación manual en el mismo proceso**

#### Patrón programación manual en el mismo modelo

Solución de diseño:

Sería con **programación manual en el propio proceso**, cuando por necesidades del proceso, se prefiere que las tareas de usuario con multi-instancia, se realicen dentro del mismo modelo de proceso, la alternativa debería pasar por “programar” el modelo de forma manual, manejando el número de tareas instancias creadas y completadas.

Por ejemplo, para una tarea multi-instancia de usuario el modelo podría quedar de la siguiente forma:

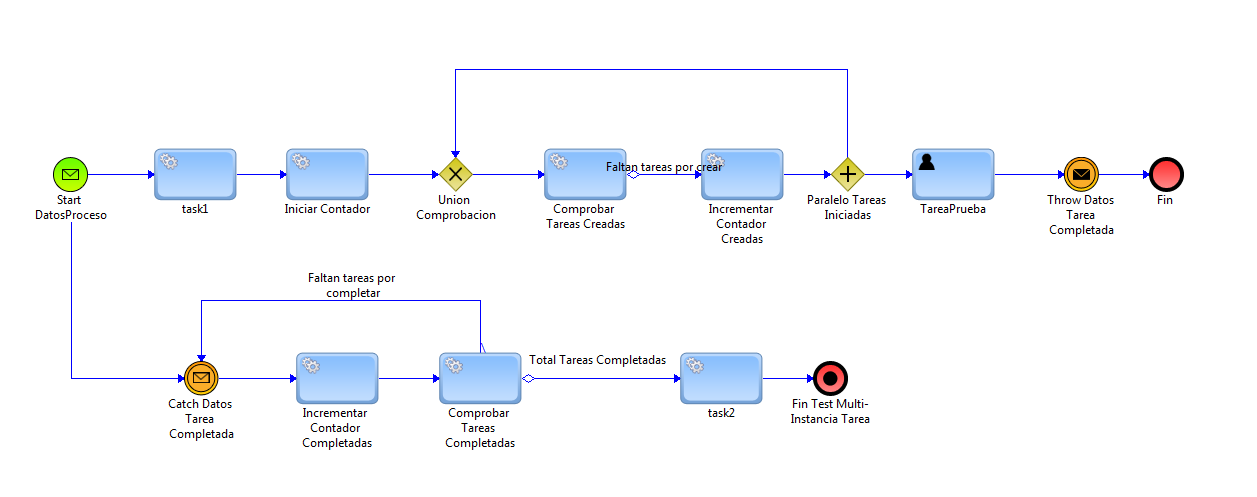


Donde el contador de tareas creadas se puede hacer con el *pipeline*, y el de completadas ha sido almacenado en base de datos de negocio, donde se graba cada vez que se completa una tarea.

El paso “Comprobar Tareas Completadas” comprobaría si faltan tareas por completar, preguntando en modelo de datos persistido por las tareas.

Otra posibilidad de “programar” la multi-instancia de tarea manual en el mismo modelo de proceso, pasaría por usar la propia publicación de eventos.

El modelo sería semejante a este:



Donde el contador de tareas completadas se controla en una rama paralela en la que sí se puede mantener el pipeline.

#### Consideraciones multi-instancia de tareas

A continuación se describen algunas consideraciones que se pueden tener en cuenta a la hora de diseñar y sobre todo implementar la solución, aparte de las ya mencionadas en los patrones de multi-instancia de procesos.

|  |  |
| --- | --- |
| **Patrón Procesos Referenciados** | **Programación manual en el propio proceso** |
| Implementación | |
| La implementación de los contadores e instancias se controla por el propio *Call Activity* | La programación es totalmente manual |

### Patrones Cancelaciones

Del mismo modo que existen eventos de negocio que como resultado dan lugar al inicio de un proceso, pueden existir otros eventos de negocio que determinen la necesidad de cancelarlo.

Aunque ese evento de cancelación podría suceder en cualquier momento de la “vida” del proceso, es importante que este aspecto se concrete en los requisitos de negocio de cada proceso.

Dependiendo de la casuística de negocio respecto de las fases del proceso en las que puede suceder ese evento, el patrón a aplicar de cancelación del mismo podría plantearse de diferentes maneras:

* ***Patrón: Cancelación en tareas manuales del proceso***

Las tareas manuales del proceso, frecuentes puntos de espera en la automatización de los mismos, pueden soportar un evento de cancelación de las mismas, determinando en ese caso la lógica de proceso a ejecutar. Este tipo de modelado, se suele dibujar ya en los modelos BPMN en ARIS de forma simillar, al que se realizará en Diseño, con un evento en la tarea de forma parecida a la siguiente:



* ***Patrón: Cancelación general del proceso***

Puede necesitarse una cancelación de forma general de un proceso en cualquier momento de la vida del mismo, con independencia del paso/tarea en el que se encuentre, se puede crear un paso específico en el proceso que se ejecute cuando suceda el evento de cancelación. Este tipo de modelado no suele venir desde ARIS, pero sí puede necesitarse su diseño. A realizar directamente en Diseño, como paso técnico del proceso.

* ***Patrón: Combinación de ambas cancelaciones***

Puede necesitrase en Diseño ambas casuisticas, disponer de la posibilidad de cancelación del proceso general y a la vez tener los eventos de cancelación en las tareas humanas.

#### Patrón: Cancelación en tareas manual del proceso

Solución de diseño:

El patrón para wM será similar al que nos encontraremos en ARIS, incorporando el evento en la tarea y las acciones que se necesiten al recibir el doumento adecuado.

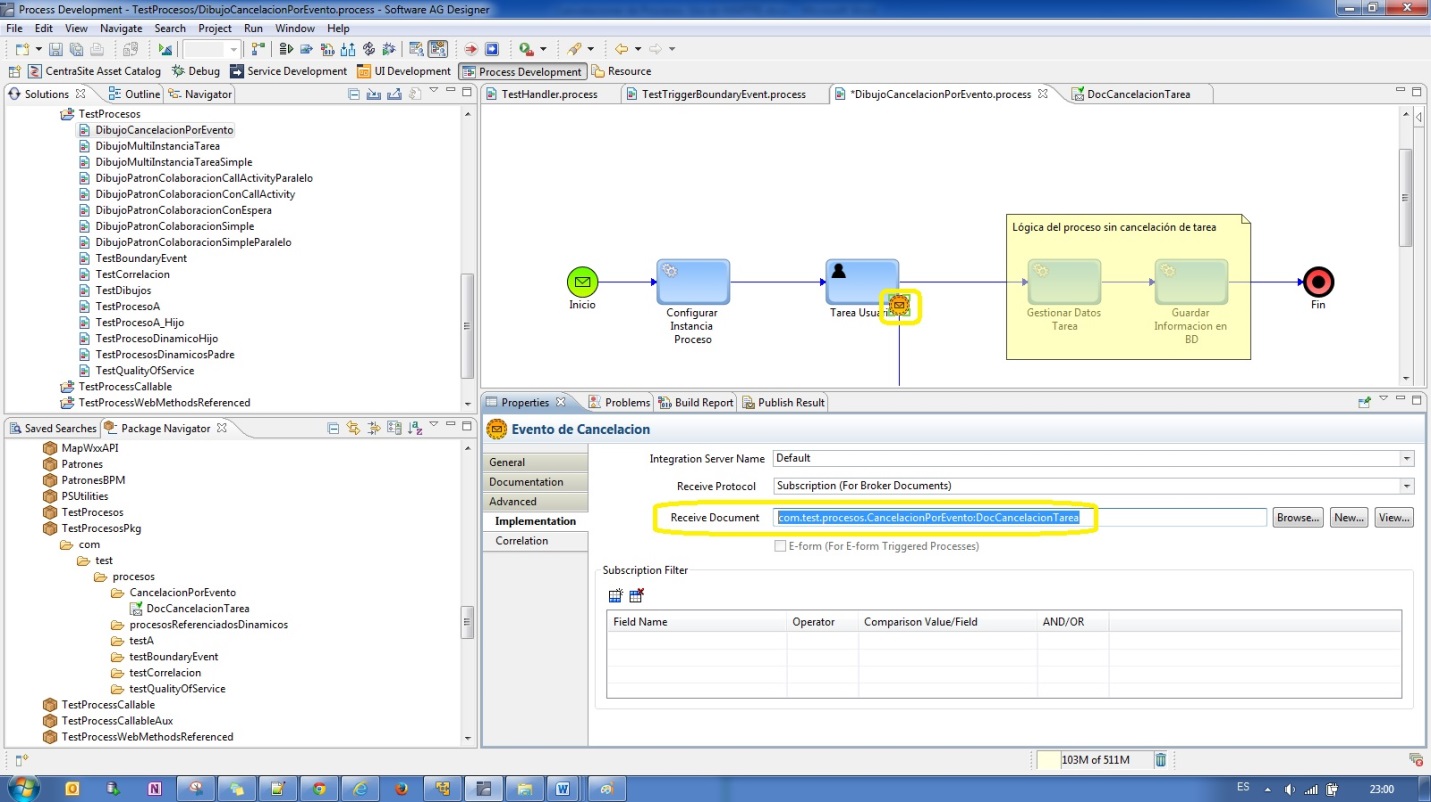
La gestión de eventos de cancelación en las tareas manuales es conveniente cuando la lógica a ejecutar tras la cancelación depende del paso o tarea del proceso en el que se encuentre, o cuando sólo esté soportada la cancelación del proceso en determinados puntos del mismo.

La cancelación de una tarea manual en concreto debe estar encapsulada en un servicio con significado de negocio que soporte ese evento específico de cancelación.

Patrón básico:

Para el evento propio de cancelación en una tarea manual, se añade a la tarea, un paso de recepción de evento (*Boundary Event*).

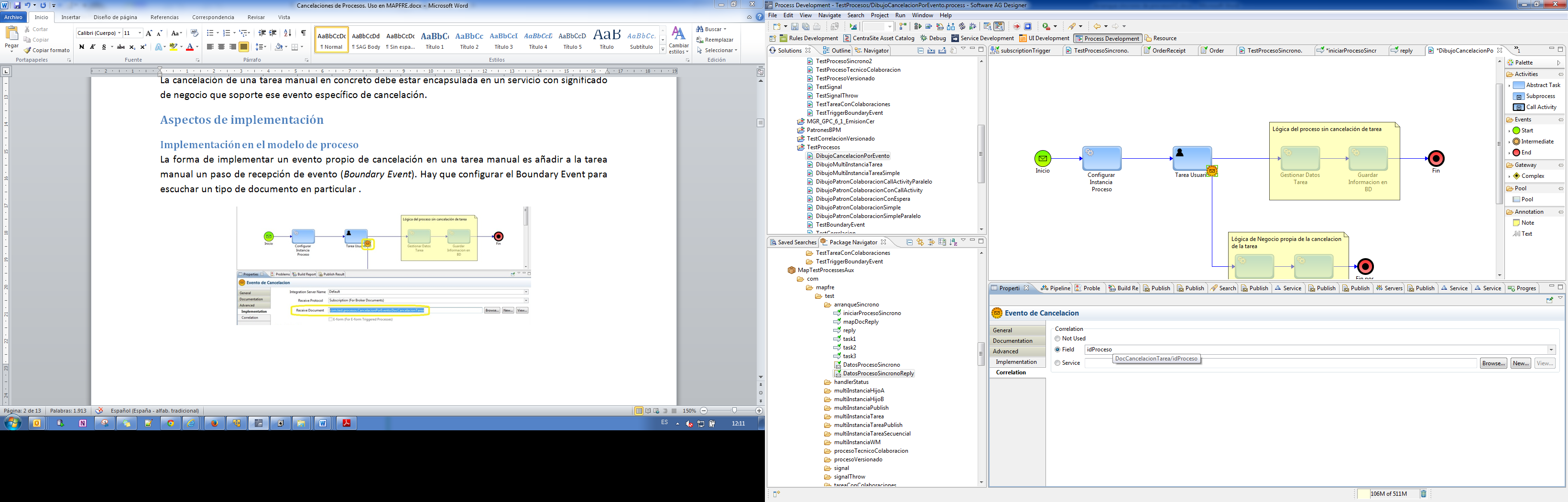
En la configuración cuando este patrón sea implementado, habrá que configurar el Boundary Event para escuchar un tipo de documento en particular y asociar la “escucha” con cada instancia de proceso en ejecución (*correlationID*).



Tal y como se recomienda en las recomendaciones de diseño actuales, la correlación entre el identificador del proceso (*ProcessInstanceID*) y un identificador de negocio (*correlationID*) se estable en un primer paso del proceso de tipo técnico (Configurar Instancia Proceso).

Además, el documento que se esta escuchando en el *Boundary Event* debe tener un campo específico en el que se informe este identificador de negocio (*correlationID*).

Una vez que se cumplen estos requisitos hay que configurar la correlación del propio *Boundary Event*. Para ello, en las propiedades del *Boundary Envent* pestaña *Correlation*, hay que marcar como *Correlation* la opción *Field* y de los campos del documento, seleccionar el campo específico que se ha comentado antes.



Volviendo al modelado del proceso, se une la salida del *Boundary Event* a los pasos encargados de realizar la logica de negocio específica para la cancelación de la tarea. De esta manera, se puede distinguir claramente el camino por el que el proceso continua si la tarea se ha completado normalmente y el camino que toma si se ha recibido un evento de cancelación de la tarea.

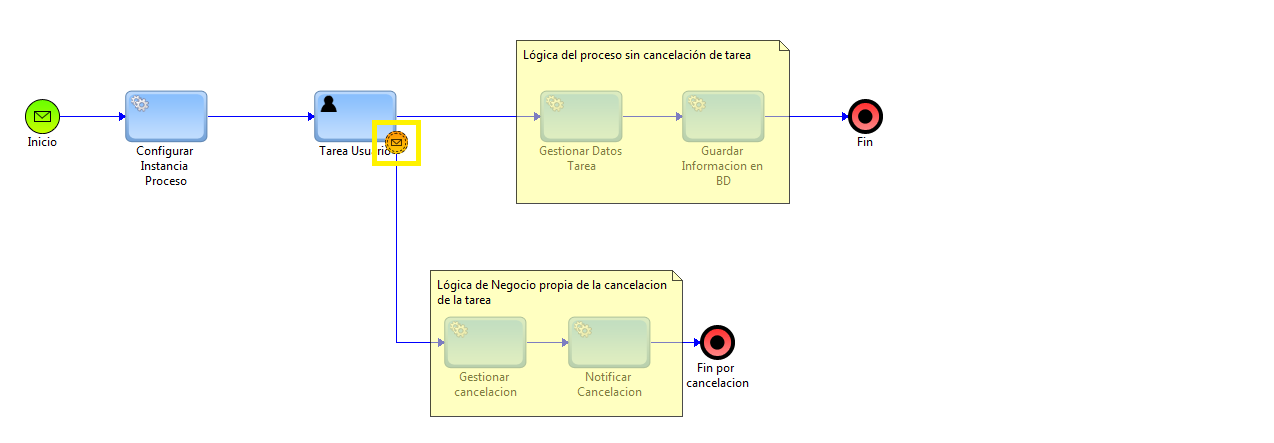


Ilustración 21 Patrón Cancelación Tarea Manual

Es importante considerar lo siguiente:

* El camino con logica de negocio para la cancelación debe terminar en un nodo de fin de proceso (*End Terminate*).
* Hay que tener en cuenta que si la lógica de cancelación no acaba en un paso de fin de proceso (*End Terminate*) se debe hacer la cancelacion de la tarea manual de forma programática.
* Sólo se podrá recibir el evento de cancelación mientras la tarea manual este activa.

La generación del evento de cancelación de la tarea se realizará ejecutando un servicio de negocio que encapsule la lógica necesaria.

Este servicio de negocio tiene que invocar al servicio *pub.publish:publish* que se encuentra dentro del paquete “*WmPublic*”, en la carpeta “*pub*”.

Al servicio *pub.publish:publish* es necesario informarle como parámetros de entrada el tipo de documento (*string documentTypeName*) y el documento (*docType document*) necesarios. El documento debe ser el mismo que se ha definido en *Boundary Event* del proceso.

#### Patrón: Cancelación general del proceso

Solución de diseño:

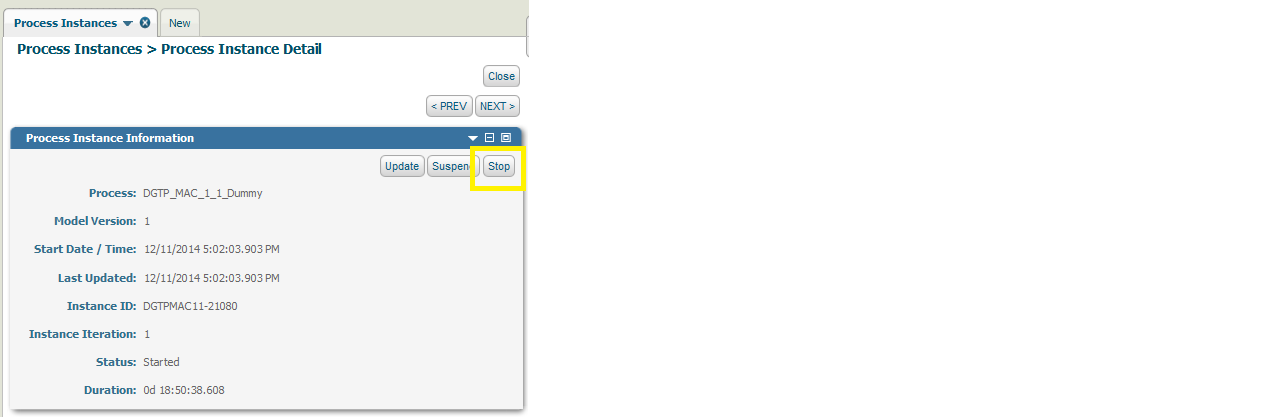
Para soportar de forma general la cancelación de un proceso en cualquier momento de la vida del mismo, con independencia del paso/tarea en el que se encuentre, se puede crear un paso específico en el proceso que se ejecute cuando suceda el evento de cancelación. Este tipo de pasos serán pasos técnicos ya que no se suelen encontrar en el modelado del análisis en ARIS.

Al igual que para el evento de inicio del proceso, la cancelación del mismo debe estar encapsulada en un servicio con significado de negocio, que reciba la información de negocio necesaria para la cancelación de dicho proceso.

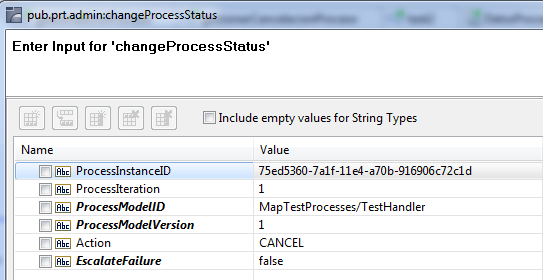
Patrón básico:

El evento de cancelación de un proceso puede desencadenarse principalmente de dos maneras:

1. Desde la consola administrativa de MWS, en la monitorización del proceso:
   * Botón “*Stop*” del “*Process Instance Information*”



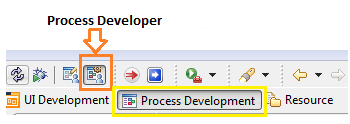
1. Mediante la invocación del servicio pub.prt.admin.changeProcessStatus, encapsulada en el servicio de negocio que soporta el evento de cancelación:



En ambos casos, el proceso iría a ejecutar un paso específico de tipo “*Cancel Handler Task*” (si este estuviera implementado), que sería la vía para ejecutar la lógica de negocio necesaria tras la cancelación.

Para poder cancelar un proceso y que este ejecute una lógica de negocio determinada es necesario en tiempo de diseño del proceso seleccionar un paso como manejador/iniciador de la cancelación (“*Cancel Handler Task*”). Este paso no necesita transición de entrada y puede tener como transición de salida tantos caminos como sea necesario por lógica de negocio.

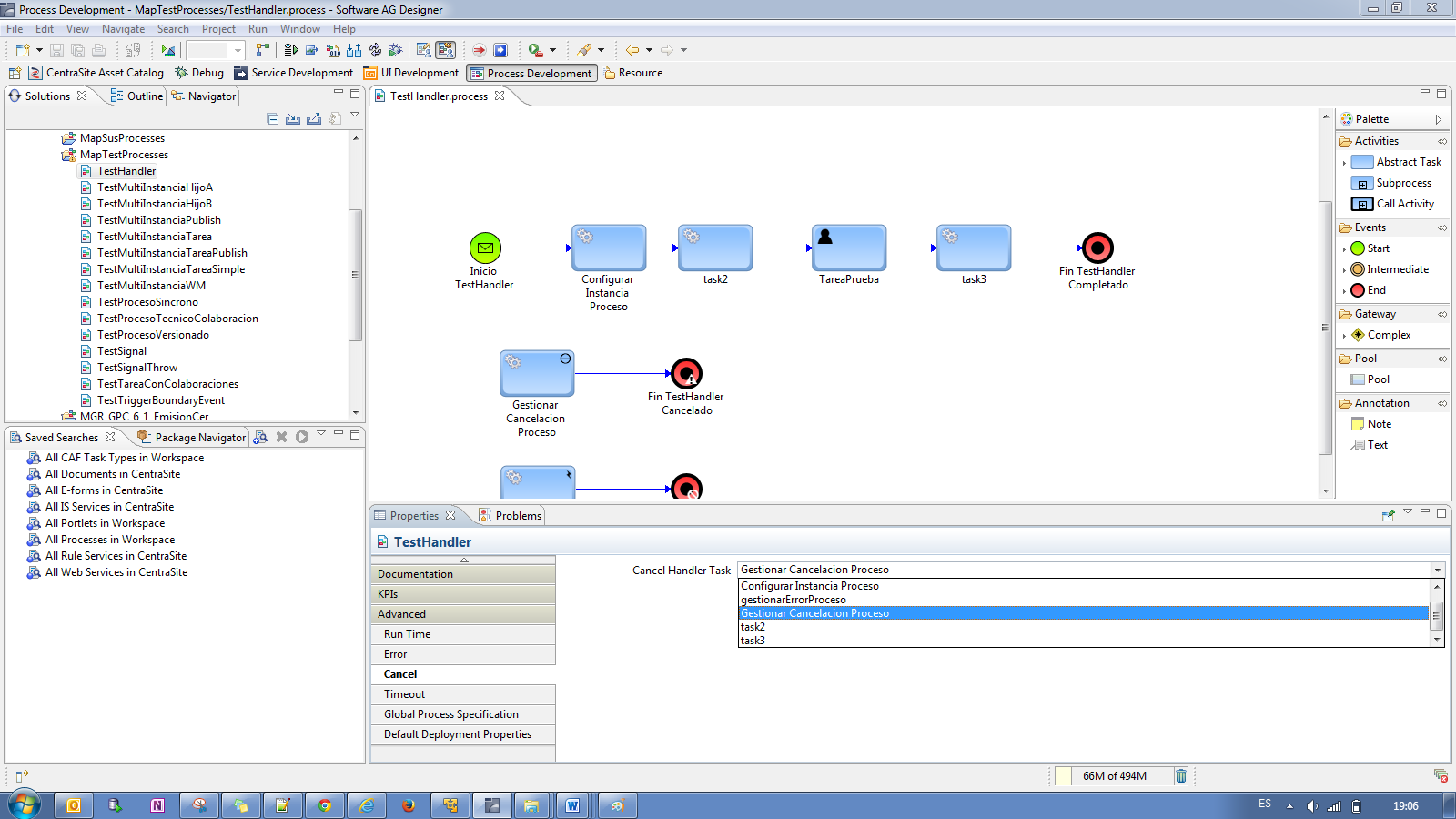
Para realizar esta acción en *Designer*, cuadno se vaya a implementar este patrón, se debe tener seleccionada la perspectiva de “*Process Development*” y la opción “*Process Developer*”.



Y en la pestaña propiedades, en la sección de propiedades avanzadas marcar la opción “*Cancel*”.

*Properties->Advanced->Cancel-> Cancel Handler Task*

Se muestra un desplegable con la etiqueta “*Cancel Handler Task*” y en él un listado de tareas del proceso que se pueden seleccionar como inicio de la lógica de cancelación del mismo. Por defecto es ninguna.



Una vez seleccionada la tarea, se distingue del resto de tareas del proceso porque se le ha añadido un símbolo con forma de círculo en la esquina superior derecha.

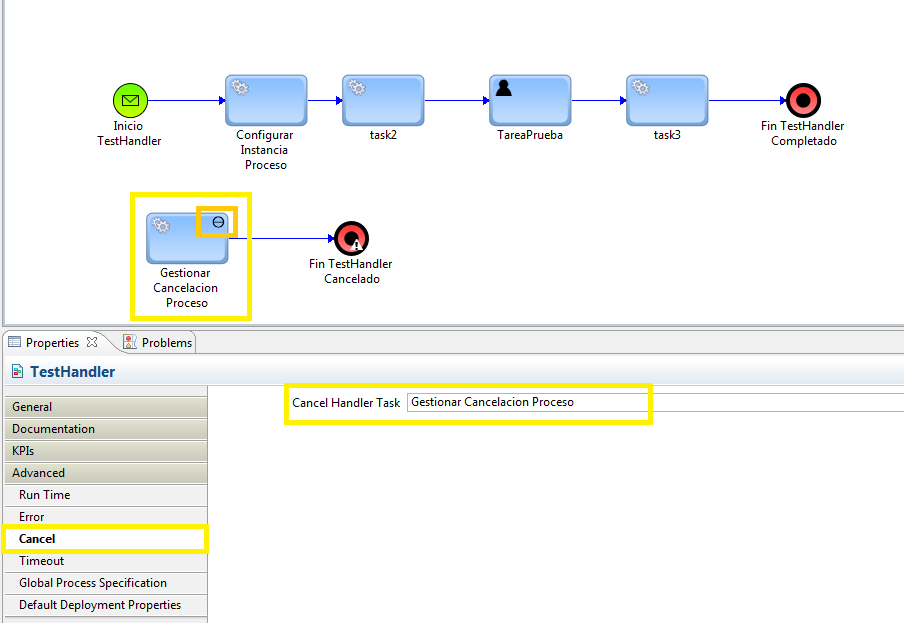


Ilustración 22 Patrón Cancelación General del proceso

El uso de un paso de finalización en el camino del “*Cancel Handler Task*” es recomendable pero no obligatorio.

Aunque se pueda seleccionar cualquier estado de finalización en el paso de fin (“*End Terminate*”) del camino del “*Cancel Handler Task*”, el estado del proceso será Cancelado (“*Stopped*”), independientemente de lo que se haya seleccionado. El paso de finalizacion debe ser de tipo “*End Terminate*” y las opciones de finalizacion se escogen en las propiedades del mismo.

La opción de terminación con estado cancelado diferencia visualmente el paso de fin de otros pasos de terminación, añadiéndole un símbolo de aviso.

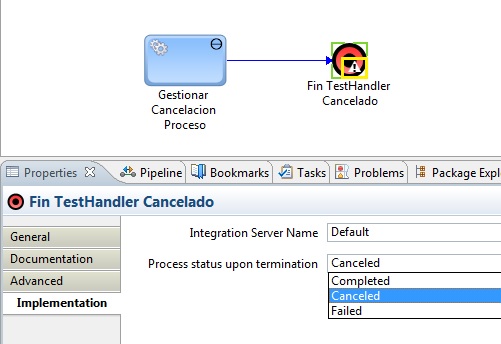


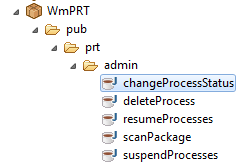
Ilustración 23 Patrón Cancelación General del proceso: Detalle estado final

Ya que, cuando se desencadena la ejecución del proceso por el camino “*Cancel Handler Task*”, el único estado posible es Cancelado, sería aconsejable elegir dicho estado de finalización en el paso de fin del proceso para que se viera reflejado en el modelado del mismo.

Como anteriormente se comenta en la cancelación de las tareas manuales, la generación del evento de cancelación se realizará ejecutando un servicio de negocio que encapsule la lógica necesaria.

Este servicio tiene que invocar al servicio pub.prt.admin.changeProcessStatus, que es un servicio del motor de procesos encargado del cambio de estado de los mismos.

El servicio changeProcessStatus se encuentra dentro del paquete “WmPRT”, en la carpeta “*admin*”.



Al servicio pub.prt.admin.changeProcessStatus es necesario informarle los siguientes datos de entrada:

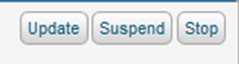
* ProcessInstanceID
* ProcessIteration
* Action : con el valor CANCEL

Ya en implementación, los datos ProcessInstanceID y ProcessIteration, al no tener significado para el negocio, se pueden obtener previamente de la llamada al servicio pub.prt.admin.changeProcessStatus, utilizando para ello el servicio del paquete “MapBPMAPI”, searchProcessInstancesSimple al que se le informara el *customID* del proceso (identificador con significado de negocio). Ver la documentación de guía del framework para más detalles.

Aunque la acción que se le debe informar al servicio pub.prt.admin:changeProcessStatus es CANCEL, se refleja como estado Stopped en el *Monitor*.

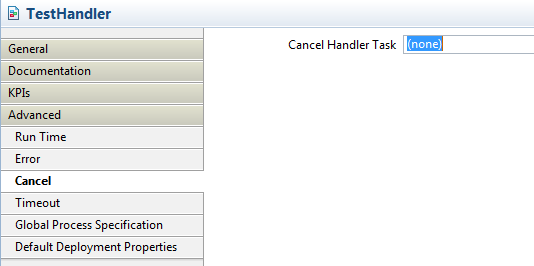
Sólo se puede cancelar un proceso cuando este está activo (estado “*Started*”). De hecho, una de las maneras comentadas anteriormente para desencadenar el evento de cancelación, el botón “*Stop*” del “*Process Instance Information*” (consola de Administración de webMethods), sólo se muestra (está habilitado) si la instancia de proceso está activa.

Es por esto que el botón “*Stop*” (al igual que “*Update*” y “*Suspend*”) desaparece una vez se ha cancelado la instancia.



Si al cancelar el proceso hubiese alguna tarea manual activa (“*active*”), esta pasa automáticamente a estado cancelado (“*canceled*”).

Si el proceso se cancela sin tener un paso seleccionado para manejar la cancelación (*Cancel Handler Task* por defecto), la instancia cambia igualmente de estado a “*Stopped*” lo único que el proceso no avanza por ningún paso.



#### Patrón: Combinación de ambas cancelaciones

En algunos casos, podría ser necesario el uso combinado de los dos enfoques anteriormente comentados. Por un lado, se soporta la cancelación de determinadas tareas manuales y el modelado dentro del proceso del camino a seguir en ese caso particular, y, por otro, se define un “*Cancel Handler Task*” general en el proceso para soportar la cancelación en cualquier punto del proceso.

Sea como fuere, tanto para lo uno como para lo otro, serán necesarios dos servicios de negocio diferentes, ya que el evento de cancelación que sucede es diferente.