*AUTOMATIZACIÓN DE PROCESOS*

*Simple Batch Manager*

*Documento de diseño*

Argentina

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Autor** | **Fecha** | **Detalle** |
| 1.0 | Leonardo Lagostena | 31/05/2013 | Primera versión |
| 1.1 | Leonardo Lagostena | 07/06/2013 | Tipos de datos y ajustes de texto |
| 1.2 | Leonardo Lagostena | 29/06/2015 | Nuevo parámetro de configuración y cambio en Descrip. Tabla de Log |
| 1.3 | Leonardo Lagostena | 27/07/2015 | Ejecución condicional y credenciales |
| 1.4 | Leonardo Lagostena | 24/08/2015 | Remoting |
| 1.5 | Leonardo Lagostena | 29/04/2016 | Revisión proceso gestión Dispatcher |
| 2.0 | Leonardo Lagostena | 15/01/2017 | Extended Platform |

Contenido

[1. Especificación de Diseño de la solución DISPATCHER 4](#_Toc357412223)

[1.1. Objetivo 4](#_Toc357412224)

[1.2. Descripción de la funcionalidad 4](#_Toc357412225)

[2. Componentes de la solución 6](#_Toc357412228)

[2.1. Web Service Dispatcher Queue 6](#_Toc357412229)

[2.1.1 Método: NewServiceRequest 6](#_Toc357412230)

[2.1.2 Método: CancelRequest 8](#_Toc357412232)

[2.2. Web Service Dispatcher Query 9](#_Toc357412234)

[2.2.1 Método: AppCatalog 9](#_Toc357412235)

[2.2.2 Método: AppQueuedServices 10](#_Toc357412236)

[2.2.3 Método: AppDispatchedServices 11](#_Toc357412237)

[2.2.4 Método: AppDispatchedService 11](#_Toc357412238)

[2.3. Windows Service Dispatcher 12](#_Toc357412239)

[2.3.1 Descripción de los componentes funcionales 12](#_Toc357412240)

[2.3.2 Esquema de Inicialización 12](#_Toc357412241)

[2.3.3 Ejecución del Servicio [Timer Event] 13](#_Toc357412242)

[2.3.4 Descripción funcionalidad GAP01, gestión AppPool 15](#_Toc357412244)

[2.3.5 Descripción funcionalidad GSC01, gestión Schedule 16](#_Toc357412245)

[2.3.6 Descripción funcionalidad GDI01, gestión Dispatcher 18](#_Toc357412246)

[2.4. Ejecución condicional 20](#_Toc357412247)

[2.4.1 Caso especial de ejecución condicional 21](#_Toc357412248)

[2.4.2 Credenciales de ejecución de los servicios 21](#_Toc357412248)

[3. Remoting 22](#_Toc357412247)

[3.1. Windows Service Agent 22](#_Toc357412229)

[3.1.1 Descripción de los componentes funcionales 23](#_Toc357412230)

[3.1.2 Esquema de inicialización 23](#_Toc357412232)

[3.1.2 Descripción funcionalidad GIO01, gestor de instancias 24](#_Toc357412232)

[4. Implementación 27](#_Toc357412247)

[4.1 Done Status 27](#_Toc357412247)

[5. Apendice 28](#_Toc357412250)

[5.1 Datos de inicialización de tablas 29](#_Toc357412251)

[5.2 Tablas del Sistema 30](#_Toc357412252)

[5.3 Diagrama de E-R 34](#_Toc357412253)

## Especificación de Diseño de la solución DISPATCHER

Objetivo

Para dar respuesta a requerimientos de distintos servicios que deben ser procesados de manera desatendida desde las aplicaciones, se conviene en desarrollar una solución que soporte este tipo de demanda. Esta solución estará preparada para brindar un conjunto de servicios prestablecidos a todas las aplicaciones que así lo soliciten y de forma centralizada.

Un ejemplo de servicios que pueden aprovechar esta solución, entre otros, puede ser:

* Reporting
* Data Transformation & Replication
* Mail delivery
* Sincronización de datos y Transacciones

Descripción de la funcionalidad

La solución abarca la implementación de dos Web Services, un servicio de Windows especializado para administrar la ejecución de assemblies o componentes específicos y un mecanismo de Feedback dedicado. Los dos servicios Web establecerán los mecanismos necesarios para solicitar, desde las distintas aplicaciones, los servicios ofrecidos y posteriormente obtener la correspondiente notificación del resultado. Estas aplicaciones deben estar previamente registradas con la solución para poder operar, ya que para instanciar estos servicios, las mismas deben proveer el código interno que la identifica y su correspondiente clave o TOKEN privado.

El primer Web Service denominado DISPATCHER\_QUEUE (DIS\_Q) estará destinado a gestionar las solicitudes de servicio. Cada aplicación que solicite una determinada prestación invocará al mismo especificando el servicio requerido, un identificador privado (opcional) de la aplicación solicitante y finalmente los parámetros que puedan corresponder. El WS DIS\_Q validará la solicitud y en caso de que la misma prospere, registrará el pedido retornando el identificador de proceso que se le haya asignado.

El segundo Web Service denominado DISPATCHER\_QUERY (DIS\_QRY) será el encargado de suministrar información tanto de catálogo (los servicios disponibles para una determinada aplicación), como también del estado de los procesos pedidos y en cola, por ejemplo, para un identificador de proceso particular, o un rango de fechas.

Estos dos WS serán los únicos mecanismos que podrán usar las aplicaciones para solicitar un servicio y conocer posteriormente el estado de dicha solicitud y su resultado. Con respecto a esto último, si bien se prevé la posibilidad de que además del estado de finalización se pueda retornar un resultado producto de la ejecución del servicio, se espera que la aplicación solicitante sea la que conozca cómo acceder por si misma al producto generado por su requerimiento. Esto es, que si el resultado de la ejecución de un servicio es un reporte, retornando su nombre de archivo, por ej., sea este suficiente dato para que la aplicación pueda localizarlo para su utilización.

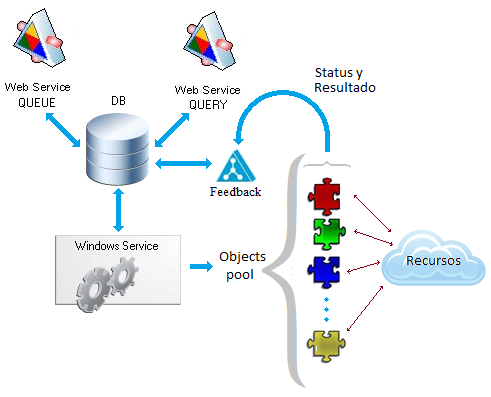
El mecanismo de Call Back (Feedback) actuará como una capa o Layer que servirá para canalizar las notificaciones desde los servicios instanciados, formando parte integral de los mismos. Por lo tanto, compartirá el ámbito de ejecución del Host y, consecuentemente, sus recursos. Este mecanismo expondrá métodos que serán los encargados de actualizar información de status y resultado en la DB de la solución, y debe estar disponible para cualquier servicio que la solución gestione. El uso de este mecanismo por parte de los servicios es mandatorio, ya que es responsabilidad de los mismos reportar información que permita seguir su ejecución así como su posterior terminación, pudiendo además, notificar un resultado si fuera necesario.

Finalmente, el servicio de Windows (WinSvr) será el encargado de revisar periódicamente la cola de solicitudes que el primer WS genera. Para proceder a su ejecución, el mismo buscará en el catálogo de servicios del sistema a los efectos de determinar cómo ejecutar el componente que, en última instancia, será el encargado de cumplir con la solicitud. El trabajo de este WinSvr finaliza cuando logra instanciar al componente elegido en un hilo de ejecución exclusivo, pasándole al nuevo objeto, el código de proceso original de la solicitud y un mecanismo para reportar su status y resultado. Otra responsabilidad del WinSvr será la de llevar un control del pool de objetos o componentes creados y activos, además de gestionar el programador de eventos.

El objeto (servicio) así instanciado, recibirá información de parametrización y clasificación. Sin embargo, es responsabilidad de cada servicio conocer cómo obtener los recursos que necesite para su operación. Esto es: acceso a datos, carpetas, archivos y cualquier otro recurso ya sea local o remoto. Por último, este servicio debe ser capaz de notificar al sistema cuando concluyó su trabajo, tanto si lo hace de forma satisfactoria o por error, a través de los mecanismos de retroalimentación previstos.

Finalmente, los mensajes emitidos en respuesta, como así también los volcados al momento de registrar la evolución y resultado de los servicios deben realizarse en inglés neutro, debiendo los mismos quedar enumerados y clasificados con un código específico en un archivo XML de configuración, que la aplicación tomará al momento de inicializarse.

A continuación se describe gráficamente los componentes de la solución:

****

## Componentes de la solución

Web Service Dispatcher Queue

Como se mencionó, este WS realiza la gestión de solicitudes de servicio. Para atender un requerimiento, la aplicación solicitante debe identificarse proporcionando un código interno (ID\_OWNER) y una clave (TOKEN), datos relacionados en la tabla DSP\_OWNER. Como todas las invocaciones los métodos de este WS deben hacerse con estos datos, es recomendable proporcionarlos una vez al instanciar la clase y verificar también que se tiene acceso (ENABLED) en DSP\_OWNER.

Métodos

NewServiceRequest

La finalidad principal del WS es registrar los pedidos de servicio en la DB de la solución, específicamente en tabla destinada a tal efecto DSP\_DISPATCHER. Esta tabla guarda las solicitudes de servicio realizadas por todas las aplicaciones que se encuentren registradas en el sistema. Cada aplicación que requiera una determinada prestación, invocará al método **NewServiceRequest** con:

* ID\_ SERVICE: Código que identifica al servicio solicitado.
* ID\_PRIVATE: (opcional) dato de la aplicación solicitante para usar como identificador interno o para filtro (por ejemplo código de usuario que hace la solicitud).
* PARAMETERS: parámetros del servicio solicitado, si los hubiere.

El WS DIS\_Q analizará la solicitud validando que el servicio requerido exista, esté disponible para la aplicación que lo está solicitando y que esté habilitado. En caso de no ser así, el WS retornará un código de error junto con una descripción clara del mismo. Por el contrario, si el requerimiento es factible, el WS registrará el pedido retornando el ID\_DISPATCHER asignado.

Para determinar si un servicio existe, este WS consultará la tabla DSP\_SERVICE\_OWNER que tiene registrada la lista de servicios (ID\_SERVICE) por aplicación solicitante (ID\_OWNER). Seguidamente, para saber si el mismo está disponible (ENABLED), debe consultarse con la tabla DSP\_SERVICE. Si alguna condición falla, como se indicó, debe retornarse una condición de error apropiada. Antes de concluir, hay que determinar si se está invocando a un servicio interno (ID\_SERVICE = 0). Si ese es el caso es necesario saber si ese servicio existe y más importante, si el mismo puede ser invocado desde una aplicación externa. Para ello, y como un caso especial, se debe analizar el campo PARAMETER, que todo servicio interno debe incluir de forma obligatoria, y leer el ID de servicio interno indicado. Con ese ID se busca en la tabla DSP\_SERVICE\_INTERNAL sobre el campo ID\_SERVICE\_INTERNAL, y si existe, se debe verificar que esté disponible (ENABLED) y que pueda ser invocado desde una aplicación externa (IS\_PUBLIC). Debe registrarse un error si alguna condición falla.

Si el pedido prospera, en la tabla DSP\_DISPATCHER se agrega un nuevo registro con el ID\_SERVICE proporcionado, junto con los correspondientes parámetros en el campo PARAMETERS. De especificarse, él o los parámetros deben definirse dentro del espacio soportado por este campo que es de tipo string y con una longitud máxima de 4000 caracteres. La forma de codificación elegida para la definición de estos parámetros dentro del campo será libre y de incumbencia sólo ente la aplicación solicitante y el servicio que lo consume. Por lo tanto, ninguno de estos WS analizará los parámetros que se proporcionen ni su validez (salvo en el caso especial indicado en el párrafo anterior). Opcionalmente, la aplicación solicitante puede proporcionar un código privado (ID\_PRIVATE) para asociarlo al código de proceso. Este código puede ser usado más tarde por dicha aplicación para determinar el origen de la solicitud (un usuario, por ej.) o para filtrar las solicitudes según un criterio particular.

Al insertarse el registro en la tabla DSP\_DISPATCHER y de forma automática, se registrará en el campo REQUESTED la fecha y la hora actual.

Se asume que una vez atendido el requerimiento de un nuevo servicio queda el mismo en cola de espera, y por lo tanto, que será ejecutado tan pronto como sea posible respetando un orden de prioridad; orden que queda establecido por su precedencia en la cola (método FIFO).

CancelRequest

Eventualmente puede surgir la necesidad de cancelar un requerimiento de servicio previamente solicitado. Para ese caso se dispondrá del método **CancelRequest**.

Con el ID\_DISPACHER proporcionado por el método anterior, el WS busca el registro apropiado en la tabla DSP\_DISPATCHER. Si existe, se inserta un nuevo registro en DSP\_DONE copiando los campos comunes, y con ID\_DONE\_STATUS con valor 10, eliminando luego el registro original de DSP\_DISPATCHER. Si el WS no encontró el registro en DSP\_DISPATCHER, significa que el mismo ya está en proceso o fue procesado. Para el primer caso, busco en la tabla DSP\_OBJ\_POOL, si lo encuentra, se actualiza el campo MAX\_TIME\_RUN con valor 0. Esto producirá su cancelación inmediata en el próximo ciclo de ejecución del Gestor del Pool de Procesos. Para el segundo caso, o si no se encuentra en el AppPool, es posible que ya haya sido procesado. Para comprobar esto, basta buscar en DSP\_DONE. De todas formas, el resultado del WS será: “Operación No Realizada”.



**Donde:**

**P1:** Inserto en DSP\_DONE campos ID\_DISPATCHER, ID\_SERVICE, ID\_OWNER, ID\_PRIVATE, PARAMETERS, REQUESTED, STARTED y ENDED con fecha actual, RESULT = Null, con código de error apropiado (10) en ID\_DONE\_STATUS.

**P2:** Con ID\_DISPATCHER busco en DSP\_OBJ\_POOL, si lo encuentro actualizo MAX\_TIME\_RUN con valor 0.

En Resumen: DIS\_Q

Métodos

**NewServiceRequest:** registra nuevos pedidos de servicio

**CancelRequest:** cancela una solicitud de servicio previa

Web Service Dispatcher Query

El segundo Web Service denominado DISPATCHER\_QUERY (DIS\_QRY) será el encargado de proveer información tanto de catálogo (los servicios disponibles para una determinada aplicación), como también del estado de los procesos pedidos (por ejemplo para un código de proceso particular, o un rango de fechas). Al igual que con el WS anterior, para atender un requerimiento, la aplicación solicitante debe identificarse proporcionando un código interno (ID\_OWNER) y una clave (TOKEN). Como todas las invocaciones a los métodos de este WS deben hacerse con estos datos, es recomendable proporcionarlos una vez al instanciar la clase y verificar también que se tiene acceso (ENABLED) en la tabla DSP\_OWNER.

Métodos

AppCatalog

Retornará la lista de Servicios disponibles para una aplicación (ID\_OWNER) específica. El método consultará las tablas DSP\_SERVICE\_OWNER, DSP\_SERVICE\_TYPE y DSP\_SERVICE, devolviendo un set de registros con los siguientes campos:

ID\_Service, Description, ID\_Service\_Type, Description, SecurityLevel, Enabled

**Donde:**

ID\_Service: Código del servicio.

Description: Descripción del servicio.

ID\_Service\_Type: Código de Tipo de Servicio.

Description: Descripción del tipo de servicio.

SecurityLevel: Código opcional que puede ser usado para determinar nivel de acceso.

Enabled: Indica si el servicio está o no habilitado para su uso.

Este método proporciona una lista dinámica de servicios disponibles para la aplicación que consulta, incluyendo información de tipo y de acceso (si la misma fue indicada). Todo lo cual, permite el armado de menús dinámicos pudiendo incluso cotejar nivel de seguridad de acceso si fuera necesario, entre otras posibilidades. No trabaja con servicios internos.

AppQueuedServices

Este método permitirá conocer la lista de servicios en cola de espera y en proceso. Esto es, la lista de pedidos realizados por una aplicación y que aún no fueron atendidos, más los que están siendo atendidos en ese momento. Para realizar esta tarea, se debe consultar la tabla DSP\_DISPATCHER especificando el ID\_OWNER y la tabla DSP\_DONE con el ID\_OWNER y el ID\_DONE\_STARUS = 2. El resultado se mostrará ordenado en forma descendente por el campo REQUESTED. Se mostrarán los siguientes campos:

ID\_Disparcher, ID\_Service, Description, ID\_Service\_Type, Description, SecurityLevel, ID\_Private, Requested, ID\_Done\_Status, Description, Started

**Donde:**

ID\_dispatcher: Código de proceso.

ID\_Service: Código del servicio.

Description: Descripción del servicio.

ID\_Service\_Type: Código de Tipo de Servicio.

Description: Descripción del tipo de servicio.

SecurityLevel: Código opcional que puede ser usado para determinar nivel de acceso.

ID\_Private: Dato opcional, que si existe, debe ser proporcionado al solicitar el servicio.

Requested: Fecha y hora en que el pedido entro en la cola de espera.

ID\_Done\_Status: Código de estado del proceso

Description: Descripción del estado del proceso según tabla.

Started: Fecha y Hora en que el pedido empezó a procesarse.

El resultado proporcionado por este WS debe ser la unión de lo recuperado de la tabla DSP\_DISPATCHER y la tabla DSP\_DONE. Para los registros de la primer tabla, el ID\_Done\_Status mostrado debe ser 1 (En cola) y Started debe estar vacío.

NOTA (\*): Se debe contemplar el caso especial para el ID\_OWNER = 0 (System), para esta clase de servicios se debe retornar información de descripción y otros atributos equivalentes a la tabla DSP\_SERVICE según lo indicado por la tabla DSP\_SERVICE\_INTERNAL (No retornar información del ID\_SERVICE = 0 de la tabla DSP\_SERVICE)

AppDispatchedServices

El método proporcionará información de servicios procesados para una determinada aplicación (ID\_OWNER) y un rango de fechas específico aplicado sobre el campo REQUESTED de la tabla DSP\_DONE. El resultado se mostrará ordenado en forma descendente por dicho campo Requested y no debe incluir filas con ID\_Done\_Status = 2. Se debe contemplar lo indicado en la NOTA (\*). Se informarán los siguientes campos:

ID\_Disparcher, ID\_Service, Description, ID\_Service\_Type, Description, SecurityLevel, ID\_Private, Requested, ID\_Done\_Status, Description, Started, Ended

**Donde:**

Idem anterior

Ended: Fecha y Hora en que el proceso Finalizó.

AppDispatchedService

Finalmente, este método proporcionará información detallada de un proceso particular (ID\_DISPATCHER) sobre la tabla DSP\_DONE (sin importar su status), controlando como siempre, que ese registro pertenezca al ID\_OWNER actual. Se debe contemplar lo indicado en la NOTA (\*). Se informarán los siguientes campos:

ID\_Disparcher, ID\_Service, Description, ID\_Service\_Type, Description, SecurityLevel, ID\_Private, Parameters, Requested, ID\_Done\_Status, Description, Started, Ended, Result

**Donde:**

*Idem anterior*

Parameters: Campo conteniendo la parametrización para el servicio invocado.

Result: Campo con el resultado de la ejecución del servicio o información de Error.

Este método retorna el registro específico completo y será el único que proporcione además, los campos usados para guardar los parámetros y el resultado (PARAMETERS, RESULT). Importante: Si el ID\_DONE\_STATUS es 2, el campo RESULT debe retornar siempre vacío.

En Resumen: DIS\_QRY

Métodos:

**AppCatalog:** lista de Servicios disponibles para una aplicación.

**AppQueuedServices:** lista de servicios en cola de espera y en proceso para una aplicación.

**AppDispatchedServices:** lista de servicios procesados para una determinada aplicación.

**AppDispatchedService:** información detallada de un proceso específico.

Windows Service Dispatcher

Este servicio representa el módulo central de la solución y será el encargado de satisfacer las solicitudes de servicio de las aplicaciones cliente, seleccionando y ejecutando las prestaciones encoladas previamente desde el WS. Parte de sus responsabilidades son: administrar la cola de procesos, ejecutar los servicios requeridos pasando los parámetros especificados, controlar el pool de objetos activos y monitorear su ejecución. También verificar la existencia de servicios que corren automáticamente y ejecutarlos si la condición requerida se satisface.

Descripción de los componentes funcionales

Este servicio tiene tres componentes que constituyen el núcleo funcional del mismo:

* Mantenimiento y gestión del pool de objetos de aplicación en ejecución [GAP01].
* Ejecución del programador de Eventos [GSC01].
* Gestión de la cola de servicios pendientes [GDI01].

Esquema de inicialización:

El servicio ejecutará las siguientes operaciones al momento de iniciar:



**Donde:**

**P1**: Efectúa la lectura de parámetros de inicialización del servicio. El servicio contará con un archivo de configuración basado en XML con parámetros de inicialización y ejecución. Además de los registros de mensajes que queden definidos, se reconocen los siguientes:

DSP\_TIMER\_INTERVAL: Expresado en segundos. Indica la frecuencia de ejecución del Timer del servicio. Default: 60. Rango: 10 - 3600.

DSP\_MAX\_OBJ\_POOL: Cantidad. Número de objetos que se pueden ejecutar en forma simultánea. Default 0 (sin límite). Rango: 0 – 9999.

DSP\_MIN\_MEMORY: En Mb. Indica la memoria mínima libre que debe tener el sistema antes de permitir la ejecución de un nuevo objeto. Default: 1024. Rango: 100 – 9999.

DSP\_BEFORE\_SHUTTING: Expresado en minutos. Indica el tiempo que el Dispatcher espera ante una falla de red, antes de auto apagarse. Default: 0 (espera indefinidamente) Rango: 0 – 1440.

Si la lectura de estos parámetros falla o si los valores de algún parámetro queda expresado fuera de los rangos admitidos, deben asumirse los valores indicados por Defalut. En este caso se debe generar un evento Audit, con información detallada: valor encontrado vs asumido.

**P2**: Con el valor determinado de DSP\_TIMER\_INTERVAL se setea el intervalo del TIMER del servicio. Esto determina la periodicidad del evento que activará la funcionalidad del módulo.

**P3**: Terminada la secuencia de carga y seteo de parámetros, debe registrarse el evento de inicio en el Event Log del sistema con código 1, indicando para cada parámetro, el valor asumido.

Cuando el servicio sea detenido (desde la consola del Windows habilitada a tal efecto) debe registrarse en el Event Log del sistema dicho evento, con código 9.

Ejecución del Servicio [Timer Event]:

Todas las operaciones desplegadas por el servicio se ejecutaran a partir de un evento de tiempo determinado por un TIMER implementado a tal fin. Disparado el evento de tiempo, una variable con scope adecuado guardará el estado de ejecución del TIMER. Esta variable puede tener dos estados posibles: Ocupado - Desocupado y actuará como un semáforo. Si el estado del TIMER es ocupado, indica que aún se encuentra en proceso un ciclo de ejecución anterior, por lo que ese evento debe finalizar sin realizar ninguna tarea. Por el contrario, si el estado es desocupado, se procederá a cambiar el estado de esa variable dándose inicio a un nuevo ciclo de ejecución.

El esquema de ejecución del TIMER es como sigue:



Descripción funcionalidad GAP01, gestión AppPool

Este proceso tiene como finalidad realizar el mantenimiento y gestión de la lista de objetos de aplicación que están en ejecución. Esta lista constituye el Objects Application Pool o AppPool y será un registro que el sistema mantendrá actualizado.

El monitoreo del estado del pool de objetos permite controlar los recursos en uso (cantidad de objetos, tiempo de proceso y memoria) así como también detectar eventos de terminación anormal en la ejecución de cada objeto, posibilitando su posterior registro.

A continuación se describe el diagrama del proceso:



**Donde:**

**P1**: Recupero todos los registros de la Tabla DSP\_OBJ\_POOL ordenado por ID\_SERVICE ASC (los internos primeros). Cada registro debe corresponder con un proceso corriendo en el servidor.

**P2**: Con el PID del proceso existente, mando mensaje de cancelación inmediata. Los procesos internos deben finalizar siempre de manera controlada y no abrupta.

**P3:** Con el PID del proceso registrado, procedo a cancelar su ejecución intentando eliminarlo. Si se trata de un proceso remoto, envío al objeto orden de cancelación.

**P4:** Registro el evento en el Event Log. De producirse un error, debe registrarse código 5, sino lleva código 4. Para ambos casos se debe completar la descripción con información de, PID, ID\_SERVICE, ID\_DISPATCHER, STARTED y MAX\_TIME\_RUN. Tomo nota del ID\_EVENT\_LOG. Busco por ID\_DISPATCHER en la tabla DSP\_DONE y actualizo ID\_DONE\_STATUS con código 6 (o 10 si MAX\_TIME\_RUN = 0) y Time Stamp en campo ENDED. Actualizo el campo RESULT (con mensaje apropiado) y el ID\_EVENT\_LOG de la tabla EVENT\_LOG, con información relativa al error.

**P5:** Con ID\_DISPATCHER elimino registro actual de DSP\_OBJ\_POOL.

**P6:** Busco por ID\_DISPATCHER en la tabla DSP\_DONE y actualizo ID\_DONE\_STATUS con código 5 y Time Stamp en campo ENDED.

Descripción funcionalidad GSC01, gestión Schedule

Parte de la responsabilidad del servicio será la de ejecutar eventos programados. Para ello se contará con un sencillo esquema de control de eventos definidos y su periodicidad, tendiente a determinar si algún evento debe ejecutarse. El evento de ejecución será el que se encuentre programado, y consistirá simplemente en la inserción automática en la tabla del Dispatcher de un nuevo registro solicitando un servicio específico. El efecto sería el mismo al generado por el Web Service Dispatcher Queue, al momento de atender un requerimiento desde una aplicación.

La evaluación del tiempo adecuado para iniciar un ciclo de ejecución dependerá de lo indicado en los campos (RUN\_INTERVAL o CONTRAB) y NEXT\_TIME\_RUN. En particular, si el último campo no está completo (Null), entonces el evento nunca se ejecutará. Será como si el mismo estuviera deshabilitado (ENABLED = False). La ejecución o no del evento surgirá de la comparación entre la fecha y Hora indicada en NEXT\_TIME\_RUN y el tiempo actual: NEXT\_TIME\_RUN < NOW. La frecuencia de evaluaión de las condiciones de ejecución depende de la frecuencia del evento de tiempo asignado al TIMER del servicio, definiéndose este valor en su inicialización.

Diagrama del proceso:



**Donde:**

**P1**: Recupera lista de Servicios habilitados (ENABLED = True) de la tabla DSP\_SERVICE\_TIMER.

**P2:** Insertar en la tabla DSP\_DISPATCHER un nuevo registro según lo registrado en DSP\_SERVICE\_TIMER (ID\_SERVICE, ID\_OWNER, ID\_PRIVATE, PARAMETERS).

**P3:** Insertar en el registro de eventos del sistema (EVENT\_LOG) un nuevo registro con código 3. En descripción indicar, con mensaje apropiado, ID\_DISPATCHER.

**P4:** Actualizar campo LAST\_TIME\_RUN de la tabla DSP\_SERVICE\_TIMER según ID\_SERVICE, ID\_OWNER, con fecha actual.

**P5:** Actualizar campo NEXT\_TIME\_RUN de la tabla DSP\_SERVICE\_TIMER según ID\_SERVICE, ID\_OWNER, con fecha actual + el Intervalo que se encuentre definido.

**P6:** Actualizar campo NEXT\_TIME\_RUN de la tabla DSP\_SERVICE\_TIMER según ID\_SERVICE, ID\_OWNER, con valor Null.

Descripción funcionalidad GDI01, gestión Dispatcher

Esta funcionalidad constituye el core de esta solución y pretende resolver la ejecución de los servicios solicitados y su posterior trazabilidad.

Resumidamente contempla la ejecución de servicios solicitados, considerando disponibilidad de recursos y registrando su evolución. Consta de una cola de procesos pendientes sucedida por una de resultados, dónde se registrará el producto de lo ejecutado, ya sea el mismo exitoso o no. Contemplando restricciones propias del servicio y recursos del sistema, dispondrá la ejecución del mismo creando el entorno adecuado.



**Donde:**

**P0:** Inserto en DSP\_DONE el nuevo registro a atender: campos ID\_DISPATCHER, ID\_SERVICE, ID\_OWNER, ID\_PRIVATE, PARAMETERS, REQUESTED, STARTED y ENDED con fecha actual, ID\_REMOTING (0 si es localhost), RESULT = Null. Por regla en la DB, al insertar se elimina solo.

**P1:** Recupero todos los registros de la Tabla DSP\_DISPATCHER en orden ascendente por el campo REQUESTED (orden FIFO). Si existen servicios internos ID\_SERVICE = 0, recupero sus atributos (ID\_SERVICE\_INTERNAL, MAX\_TIME\_RUN, SINGLE\_EXEC, ASSEMBLY\_FILE).

**P2:** Si el parámetro de inicialización del servicio DSP\_MAX\_OBJ\_POOL es distinto de 0, cuento el número de objetos actualmente instanciados según lo indique la tabla DSP\_OBJ\_POOL y si no se alcanzó el límite indicado, puedo continuar. Seguidamente analizo el parámetro de inicialización DSP\_MIN\_MEMORY comparando con el nivel de memoria ocupada actual. Además, si el componente es remoto, debe requerirse el mismo análisis al Agente correspondiente.

**P3:** Insertar en el registro de eventos del sistema (EVENT\_LOG) un nuevo registro con código 2. En descripción indicar, con mensaje apropiado, ID\_DISPATCHER, número de objetos y memoria RAM libre actual. Si la falta de algún recurso corresponde al componente remoto, debe indicarse. Actualizo en DSP\_DONE campos ENDED con fecha actual, RESULT: indicar, con mensaje apropiado y ID\_DONE\_STATUS = 14.

**P4:** Busco en DSP\_SERVICE y DSP\_SERVICE\_OWNER por ID\_SERVICE y ID\_OWNER si existe el servicio y está habilitado (ENABLED True). Tomo valores SINGLE\_EXEC, X86 y ID\_PARENT\_SERVICE. Si se trata de un servicio interno (ID\_SERVICE = 0), además busco por ID\_SERVICE\_INTERNAL en la tabla DSP\_SERVICE\_INTERNAL, verifico si está ENABLED.

**P5:** Actualizo DSP\_DONE con código de error apropiado (7, 8, 9 o 13).

**P6:** Busco en la tabla DSP\_OBJ\_POOL si hay un registro con ese ID\_SERVICE, o si se trata de un servicio interno ID\_SERVICE\_INTERNAL.

**P7:** Insertar en el registro de eventos del sistema (EVENT\_LOG) un nuevo registro con código 6. En descripción indicar, con mensaje apropiado, ID\_DISPATCHER, ID\_SERVICE, DSP\_SERVICE.DESCRIPTION (sólo los primeros 100 caracteres). Actualizar en DSP\_DONE campos ENDED con fecha actual, RESULT = Null y ID\_DONE\_STATUS = 11.

**P8:** De DSP\_SERVICE (1) y DSP\_DONE (2) recupero datos para ejecutar el servicio. De (1): ASSEMBLY\_FILE, ASSEMBLY\_PATH, MAX\_TIME\_RUN, X86. De (2): ID\_PRIVATE, PARAMETERS.

**P9:** Instancio servicio en Hilo de ejecución exclusivo, paso parámetros (ID\_DISPATCHER, PARAMETRS). Si es un servicio interno, paso además referencia de la base de datos propia del sistema, para su adecuado acceso y su propio ID\_SERVICE\_INTERNAL, para que recupere información de configuración desde el campo CONFIG de la tabla DSP\_SERVICE\_INTERNAL, si fuera necesario. Si se trata de un componente x86 (tabla DSP\_SERVICE), armo entorno adecuado, de lo contrario se asume un entorno de 64bits. Si el componente tiene credenciales específicas (campos DOMAIN, USER, PASSWORD de la tabla DSP\_SERVICE), se usan las mismas al instanciar.

**P10:** Actualizo DSP\_DONE campos ID\_REMOTING (0 si es localhost). Agrego registro en DSP\_OBJ\_POOL, campos: ID\_DISPATCHER, ID\_SERVICE, PID, STARTED = fecha actual y MAX\_TIME\_RUN según valor de DSP\_SERVICE (o DSP\_SERVICE\_INTERNAL si es un servicio interno, en ese caso también completo el campo ID\_SERVICE\_INTERNAL). Nota Importante: Si el campo MAX\_TIME\_RUN de DSP\_SERVICE tiene el valor 0 significa que no se debe controlar límite de tiempo, por lo tanto debe guardarse con Null en la tabla DSP\_OBJ\_POOL al momento de insertar.

**P11:** Verifico si ID\_PARENT\_SERVICE en DSP\_SERVICE tiene un ID de servicio asignado. Si tiene un ID válido, busco en la tabla DSP\_DONE la primer ocurrencia en orden temporal descendente (más reciente a más viejo), pero ignorando registros con ID\_DONE\_STATUS <> 1, 2, 3, 4, 5, 6, 7 (Al momento: 8, 9, 10, 11, 12, 13, 14). Si se encuentra una coincidencia, y la misma es ID\_DONE\_STATUS = 3, entonces se procede con la correspondiente ejecución del servicio. En cambio si el ID\_DONE\_STATUS = (1, 2, 4, 5, 6, 7) o no existe registro precedente, entonces debe cancelar la ejecución automáticamente.

**P12:** Actualizo en DSP\_DONE campos ENDED con fecha actual, RESULT: indicar, con mensaje apropiado, ID\_PARENT\_SERVICE, Parent Service Status (Código, Descripción) + Fecha de ese status.

**P13:** Si ID\_SERVICE = 0 se trata de un servicio interno, en ese caso se instancia local en directorio exclusivo. Sino, busco en la tabla ID\_REMOTING si el servicio (ID\_SERVICE) debe invocarse en un equipo remoto. Si no existe, entonces el Servicio es local. Pero si el ID\_SERVICE existe, entonces preparo componente interno para invocación remota. Tomo ID\_REMOTING.

Ejecución condicional:

Dispatcher implementa un sencillo mecanismo de ejecución condicional de componentes, a los efectos de establecer alguna relación de dependencia jerárquica entre ellos. La idea es que se pueda determinar precedencia entre componentes para asegurar que, si se produce una condición no satisfactoria del componente referido (como una terminación anormal, por ej.), el componente dependiente no ejecute. Para este motivo se usa el campo ID\_PARENT\_SERVICE del registro del servicio actual en DSP\_SERVICE. Ese campo podrá referir a un ID de la misma tabla, estableciendo así la relación de dependencia. De esta manera, es posible encadenar servicios estableciendo una relación “padre – hijo” simple.

Cuando ese campo tiene un ID válido, se busca en la tabla DSP\_DONE la primer ocurrencia en orden temporal descendente (más reciente a más viejo), pero ignorando registros con ID\_DONE\_STATUS **<>** 1, 2, 3, 4, 5, 6, 7 (Al momento: 8, 9, 10, 11, 12). Si se encuentra una coincidencia, y la misma es ID\_DONE\_STATUS = 3, entonces se procede con la correspondiente ejecución del servicio actual. En cambio si el ID\_DONE\_STATUS = (1, 2, 4, 5, 6, 7), entonces debe cancelar la ejecución con ID\_DONE\_STATUS = 12 [Canceled, Parent service not ready].

Caso especial de ejecución condicional:

A través del campo ID\_PARENT\_SERVICE es posible referir a cualquier ID válido (existente) incluso ese ID puede ser el mismo del ID\_SERVICE actual (ID\_SERVICE = ID\_PARENT\_SERVICE). En ese caso especial, se interpreta que cuando el mismo servicio tenga una terminación anormal (distinta de ID\_DONE\_STATUS = 3), entonces no volverá a ejecutarse (al menos automáticamente).

Dado que esta condición impediría reanudar la ejecución del servicio nuevamente, ya que la condición requerida nunca se cumpliría, entonces se prevé el cambio del ID\_PARENT\_SERVICE (con valor = ID\_SERVICE) por el valor 0. Ese valor le indicará al Dispatcher que excepcionalmente no debe verificar ejecución satisfactoria previa, reemplazando luego de salteada la verificación, el cero en ID\_PARENT\_SERVICE por el ID\_SERVICE actual. Eso garantiza que la siguiente ejecución sea normal, volviendo a verificarse que la ejecución precedente sea exitosa como requisito previo.

Credenciales de ejecución de los servicios:

En casos en que se deba correr el componente de servicio bajo credenciales específicas, se prevé el uso de los campos DOMAIN, USER, PASSWORD. Cuando se completan estos campos opcionales, el Dispatcher intenta levantar el servicio bajo las credenciales indicadas. Si no se especifican esos datos, el Dispatcher usa las mismas credenciales que obtuvo al momento de su instanciación como servicio de Windows.

El comportamiento descripto refiere tanto a servicios que deben ser instanciados en modo local como remoto, siendo en este último caso el Agente correspondiente, quien realice la operación. Consecuentemente, si el servicio es remoto, las credenciales deberán ser válidas en el entorno de ejecución designado.

## Remoting

Para casos en que la ejecución de un componente no se pueda efectuar en el ámbito local, se prevé la posibilidad de su ejecución remota. Para ello se brinda el uso de un Agente que correrá en el equipo remoto, y tendrá como finalidad recibir y administrar todas las solicitudes de ejecución y seguimiento que se puedan realizar desde el servicio central Dispatcher.

Windows Service Agent:

Conceptualmente, el Agente será una versión funcional del mismo Dispatcher, pero con un alcance reducido o acotado a un conjunto de actividades específicas, estando además subordinado al servicio principal (Dispatcher). Como su funcionamiento no será autónomo, los mecanismos de instanciación de objetos se activaran exclusivamente bajo demanda, salvo aquellos procesos de mantenimiento interno que requieran una ejecución periódica específica. Consecuentemente este componente será también un servicio de Windows, pero a diferencia del principal, no tendrá acceso a ninguna base de datos, ni a otros recursos distintos a los propios de su instalación. Todos los eventos producidos serán reportados al Dispatcher a través de los componentes que el mismo agente ejecute, y de un conjunto mínimo de APIs que este servicio expondrá, y que el componente de ejecución local al servicio central podrá acceder. Sin embargo, compartirá con el Dispatcher original, los mecanismos de inicialización según parámetros de configuración, realizando gestión de memoria y de objetos instanciados de manera similar.

Por lo expuesto, el Agente no tendrá un Schedule ni un gestor de pool de objetos (no podrá por ejemplo, controlar el evento de time out, que queda como responsabilidad del Servicio principal), y la gestión interna quedará reducida al conteo de objetos activos y límite de memoria disponible propia. Adicionalmente, con la idea de prever en el futuro la posibilidad de balanceo de carga, es deseable que este componente cuente con mecanismo simple de censo de los recursos, como uso de CPU y de algún otro medio sensible como accesos de I/O y espacio en disco.

Para implementar esta funcionalidad, la solución dispondrá de una tabla interna denominada DSP\_REMOTING, que contendrá información relativa al servidor que efectuará la instanciación del servicio. El Dispatcher buscará el ID\_SERVICE en esa tabla para determinar si se trata de un servicio de instanciación remota. Si encuentra ese ID, procede a levantar un módulo interno de Remoting, que será el encargado de tomar los parámetros de la tabla DSP\_REMOTING y conectar con el Agente situado en el servidor indicado, pasando la información necesaria para la instanciación remota del componente requerido. Este componente permanecerá activo durante todo el ciclo de vida del componente remoto, y será el encargado de administrar toda la comunicación y respuestas de dicho componente y del Agente. A los efectos prácticos definimos que asume **su identidad**, y actuará como un espejo local de su equivalente remoto, permitiendo al Dispatcher obtener y registrar información de auditoría y control, como si se tratara de un componente local standard. Este comportamiento es fundamental para permitir el control consistente del AppPool, sobre todo en lo referente a los componentes de **ejecución única**. Es importante notar además, que no puede coexistir un mismo servicio como local y remoto a la vez.

La arquitectura del Agente así dispuesta, junto con el componente local de comunicación remota, intenta ocultar la complejidad de gestión de componentes remotos, a la vez que evita la necesidad de una adaptación profunda de la solución para contemplar esta funcionalidad.

Descripción de los componentes funcionales

A diferencia del servicio Dispatcher principal, el Agente tendrá un único núcleo funcional significativo, que es el gestor de instancia de objetos [GIO01]. El control del pool de objetos debe limitarse a lo efectivamente activo en memoria y, por lo tanto, debe compararse al momento de instanciar un nuevo objeto a los efectos de respetar el parámetro DSP\_MAX\_OBJ\_POOL propio del Agente. Ante la eventualidad de una caída del servicio o del mismo servidor, la consecuencia es que el Dispatcher observaría un time out en la respuesta del o los componentes remotos, por lo que procedería a cancelarlos siguiendo el protocolo usual. Este resultado es considerado consistente con lo ocurrido, y por lo tanto suficiente para el registro y control de la incidencia.

Esquema de inicialización:

Al igual que el servicio Dispatcher, el Agente ejecutará las siguientes operaciones al momento de iniciar:



**Donde:**

**P1**: Efectúa la lectura de parámetros de inicialización del servicio. El servicio contará con un archivo de configuración basado en XML con parámetros de inicialización y ejecución. Además de los registros de mensajes que queden definidos, se reconocen los siguientes:

DSP\_TIMER\_INTERVAL: Expresado en segundos. Indica la frecuencia de ejecución del Timer del servicio. Default: 60. Rango: 10 - 3600.

DSP\_MAX\_OBJ\_POOL: Cantidad. Número de objetos que se pueden ejecutar en forma simultánea. Default 0 (sin límite). Rango: 0 – 9999.

DSP\_MIN\_MEMORY: En Mb. Indica la memoria mínima libre que debe tener el sistema antes de permitir la ejecución de un nuevo objeto. Default: 1024. Rango: 100 – 9999.

DSP\_BEFORE\_SHUTTING: Expresado en minutos. Indica el tiempo que el Dispatcher espera ante una falla de red, antes de auto apagarse. Default: 0 (espera indefinidamente) Rango: 0 – 1440.

Si la lectura de estos parámetros falla o si los valores de algún parámetro queda expresado fuera de los rangos admitidos, deben asumirse los valores indicados por Defalut. En este caso se debe generar un evento Audit, con información detallada: valor encontrado vs asumido.

**P2**: Con el valor determinado de DSP\_TIMER\_INTERVAL se setea el intervalo del TIMER del servicio. Esto determina la periodicidad del evento que activará la funcionalidad del módulo, que se contempla al solo efecto de realizar operaciones periódicas como puede ser el censo de recursos disponibles, entre otros.

**P3**: Terminada la secuencia de carga y seteo de parámetros, debe registrarse el evento de inicio en el Event Log del sistema con código 1, indicando para cada parámetro, el valor asumido.

Cuando el servicio sea detenido (desde la consola del Windows habilitada a tal efecto) debe registrarse en el Event Log del sistema dicho evento, con código 9.

NOTA IMPORTANTE: Dado que el agente no tiene acceso a un mecanismo de persistencia de datos, toda la información que normalmente quedaría registrada en el esquema de LOG del Dispatcher, debería registrarse ahora en un archivo de texto destinado a tal fin, dentro del directorio de instalación del Agente, y con un mecanismo de crecimiento controlado o limitado.

Descripción funcionalidad GIO01, gestor de instancias

Esta funcionalidad constituye el core de la prestación de este proceso y pretende resolver la ejecución de los servicios solicitados y su posterior trazabilidad.

En pocas palabras, la funcionalidad del Agente está conformada por dos tareas principales. Por un lado su gestión interna para propio control, así como control de recursos disponibles y servicios ya en ejecución. Y por otro lado el gestor de instancias, encargado de levantar los servicios en el entorno de ejecución requerido (por ejemplo 32 o 64bits) y de disponibilizar los recursos del servidor necesarios para ello. La primer tarea resulta trivial y por ende, no requiere una descripción detalla a los fines de este documento. La segunda empero, constituye la parte más importante del funcionamiento, por lo que se describe en el siguiente diagrama.



**Donde:**

**P1**: Ante el pedido de una nueva instanciación, se debe verificar que la cantidad de objetos activos sumado a este nuevo pedido, no supere el límite establecido por el parámetro de configuración DSP\_MAX\_OBJ\_POOL. De la misma manera se controla previamente el estado de la memoria interna disponible, según lo indicado por el parámetro DSP\_MIN\_MEMORY.

**P2**: Se notifica al componente llamador y se cancela la operación.

**P3:** Instancio servicio en Hilo de ejecución exclusivo, paso parámetros (ID\_DISPATCHER, PARAMETRS). Si se trata de un componente x86, armo entorno adecuado. Si el componente tiene credenciales específicas (campos DOMAIN, USER, PASSWORD informados), se usan las mismas al instanciar.

**P4:** Agrego registro en POOL de Objetos interno, incremento contador. Notifico Operación con éxito al componente llamador.

**P5**: Se notifica al componente llamador informando el error con el mayor detalle posible, y se cancela la operación.

Es importante remarcar que el funcionamiento de este esquema de ejecución remota de componentes, basa su éxito en la implementación de una comunicación bidireccional ente el componente local y el remoto. De forma local, el Dispatcher instancia un componente espejo del componente que el Agente instancia en forma remota. Si bien el componente espejo local al Dispatcher es genérico, y es el mismo para todos los componentes a ejecutarse remotamente, el componente local genérico debe asumir la identidad del componente remoto y reflejar su evolución de manera completa. En la práctica actúa como un “handler” del componente remoto. Por tal motivo, es de singular relevancia la calidad del canal de comunicación entre componente local y remoto. Si la comunicación se cae ente los mismos, ambos componentes usan el parámetro MAX\_RESPONSE\_TIME de la tabla DSP\_REMOTING para establecer un tiempo de tolerancia ante la interrupción de la comunicación. Si en ese tiempo el enlace no se restablece, ambos componentes son Cancelados en su ejecución, reportando el evento según los protocolos establecidos.

## Implementación

Para que los Servicios instanciados cuenten con un mecanismo estándar para actualizar información de status y resultado en la DB de la solución, es necesario utilizar una interface específica. Los servicios deberán implementar una interfaz llamada Batch. Esta interfaz posee un único método llamado Submit, el cual recibe un valor PARAMETER como argumento y devuelve otro string, que se guardará en RESULT como se explica más adelante.

Opcionalmente podrá implementarse otra interfaz llamada BatchCancelable, que extiende de Batch. Dicha interfaz tiene un método llamado Cancel, cuya implementación debería indicar al proceso principal que debe finalizar la ejecución. El mismo servicio puede crear múltiples hilos (si fuera requerido) propiciando la ejecución en paralelo, pero se descargará todo el conjunto de la memoria cuando el hilo principal finalice.

DoneStatus

Cada servicio debe retornar una única vez su resultado siendo éste, por lo tanto, el definitivo. Por consiguiente, hay dos posibles estados de finalización: por Éxito o Fracaso, según lance o no una excepción. Si el servicio lanzó una excepción o envió un valor de retorno, se procede a actualizar el campo ID\_STATUS\_DONE con código 3 o 4 según sea éxito o excepción.

Al finalizar y reportar su status, si se pasa información en el campo RESULT, se debe pisar cualquiera preexistente. Pero si no se proporciona, se debe insertar el valor Null, para limpiar cualquier contenido previo que el campo pudiera tener.

## APENDICE

## Datos de inicialización para las siguientes tablas:

**DSP\_DONE\_STATUS**

1 In Queue (Este estado es inherente sólo a los registros de la tabla DSP\_Disparcher) InQ

2 In Process InProc

3 Completed successfully Ok

4 Completed with errors (El mismo Servicio lo reporta, Result puede tener más info) NoOK

5 Canceled, fatal error (El servicio finalizó sin actualizar su estado, posible falla) X-FErr

6 Canceled, service timeout (Cancelado por superar el tiempo máximo de ejecución) X-TMO

7 Canceled, Internal error (Falla al instanciar el servicio, o tuvo un error al ejecutar) X-IErr

8 Service disabled SvcOff

9 Service unknown SvcUnK

10 Canceled, user request X-UserR

11 Canceled, service already running X-SvcRun

12 Canceled, Parent service not ready X-PSvcNR

13 Canceled, invalid Owner X-XOwn

14 Canceled, not enough resources X-NoRes

**DSP\_SERVICE\_TYPE**

1 Report RPT

2 Data Transformation & Replication (ETL) DTS

3 Mail Delivery & Email Spool MAIL

4 Transaction & Synchronization TRX

5 Business Logic BLog

6 Interop InTOP

7 Generic ANY

8 Internal SELF

**DSP\_OWNER**

0, [System], [69i57j0l5], 1

**DSP\_SERVICE**

0, [Internal Service Broker], 1, (CurDate), 8, [NONE], [NONE], 0, 0, 1, NULL, NULL, NULL, 0

**DSP\_REMOTING**

0, [localhost], 0, [127.0.0.1], [localhost], 0, 1

**DSP\_EVENT**

0 Unspecified

1 Dispatcher Service Started

2 Service pool or memory full, not enough resources

3 New scheduled service in queue

4 Process killed, timeout

5 Error killing running process

6 Service already running

7 Audit

8 [Reserved]

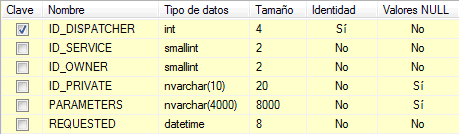
9 Dispatcher Service Ended

**DSP\_SERVICE\_OWNER**

0, 0, 0

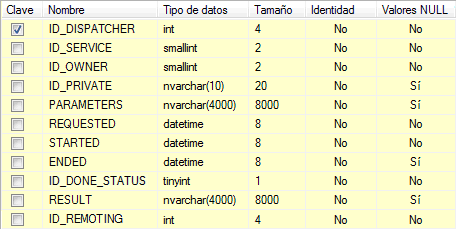
## Tablas del Sistema

**DSP\_DISPATCHER**

****

Observaciones: **REQUESTED** debe tener valor por default GetDate()

**DSP\_DONE**

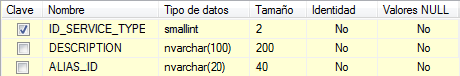
****

Observaciones: TRIGGER al insertar, elimina en DSP\_DISPATCHER según **ID\_DISPATCHER**.

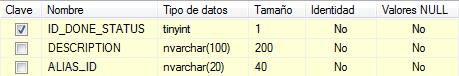
**STARTED** debe tener valor por default GetDate()

**ID\_DONE\_STATUS** debe tener valor por default 2

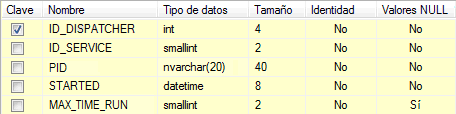
**DSP\_SERVICE\_TYPE**



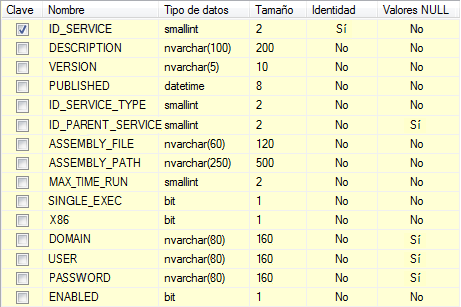
**DSP\_DONE\_STATUS**

****

**DSP\_OBJ\_POOL**



**DSP\_SERVICE**

****

Observaciones: **PUBLISHED** debe tener valor por default GetDate()

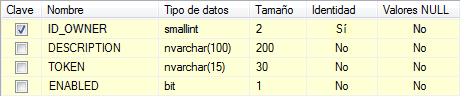
**ID\_SERVICE\_TYPE** debe tener valor por default 1

**MAX\_TIME\_RUN** debe tener valor por default 0 (expresado en segundos)

**SINGLE\_EXEC** y **ENABLED** deben tener valor por default True

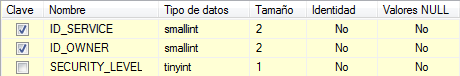
**X86** debe tener valor por default False

**DSP\_OWNER**



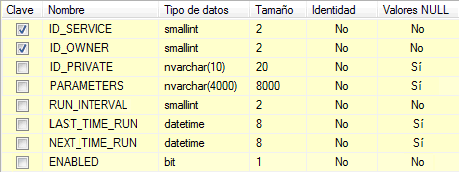
Observaciones: **ENABLED** debe tener valor por default True

**DSP\_SERVICE\_OWNER**



Observaciones: **SECURITY\_LEVEL** debe tener valor por default 0

**DSP\_SERVICE\_TIMER**

****

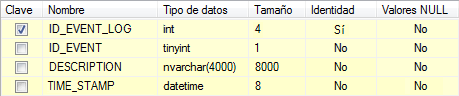
Observaciones: **RUN\_INTERVAL** debe tener valor por default 0

**ENABLED** debe tener valor por default True

**DSP\_EVENT**

****

**DSP\_EVENT\_LOG**

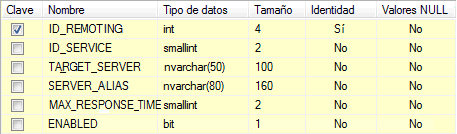
****

Observaciones: **ID\_EVENT** debe tener valor por default 0

**DESCRIPTION** debe tener valor por default “”

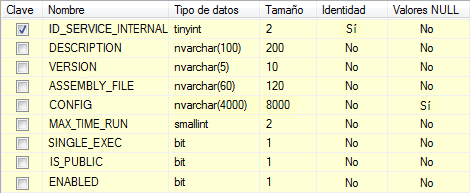
**TIME\_STAMP** debe tener valor por default GetDate()

**DSP\_REMOTING**



NOTA: Para DSP\_SERVICE\_TYPE y DSP\_DONE\_STATUS, el campo ALIAS\_ID refiere a un nombre nemotécnico o abreviatura del campo DESCRIPTION de sus respectivas tablas (sin espacios).

**DSP\_SERVICE\_INTERNAL**



Observaciones: **MAX\_TIME\_RUN** debe tener valor por default 0 (expresado en segundos)

**SINGLE\_EXEC** y **ENABLED** deben tener valor por default True

**IS\_PUBLIC** debe tener valor por default False

## Diagrama E-R

