ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
GRADO EN INGENIERÍA ELECTROMECÁNICA

DESARROLLO DE UNA NUEVA FUNCIONALIDAD INTEGRADA EN LA APLICACIÓN PATIO PRO DE 4TPM

Autor: Jean Nadal
Director: Eric Thierry
Madrid, Agosto 2018

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Jean Nadal DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Desarrollo de una nueva funcionalidad integrada en la aplicación PATIO PRO de 4TPM , que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

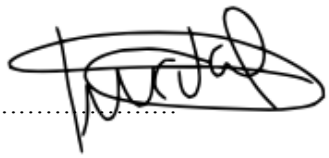
a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.
La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 23 de Agosto de 2018

ACEPTA



Fdo…… Jean Nadal

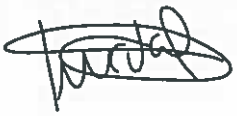
Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo de una nueva funcionalidad integrada en la aplicación PATIO Pro de 4TPM

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2017-2018 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

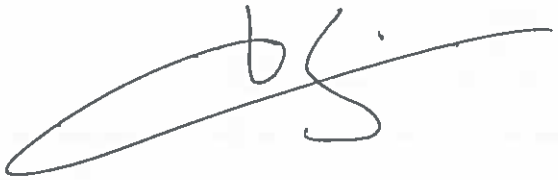
Fdo.: Jean Nadal Fecha: 23 /08 / 2018



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Eric Thierry Fecha: 28 / 08 / 2018



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
GRADO EN INGENIERÍA ELECTROMECÁNICA

DESARROLLO DE UNA NUEVA FUNCIONALIDAD INTEGRADA EN LA APLICACIÓN PATIO PRO DE 4TPM

Autor: Jean Nadal
Director: Eric Thierry
Madrid, Agosto 2018

AGRADECIMIENTOS

Quiero agradecer en primer lugar a todos los compañeros de 4TPM, por su cálido recibimiento y la atención recibida a lo largo de los 5 meses de prácticas. Me gustaría agradecer particularmente a Eric Degagny y a Eric Thierry, por permitirme realizar las prácticas en 4TPM lo que ha sido una experiencia formidable de un punto de vista laboral y personal, al igual que a John Gallet y Frederic Marianne por toda la ayuda recibida a lo largo de todo el proyecto. En segundo lugar, a Aurelio García Cerrada por su labor de tutoría y seguimiento a lo largo de este año. Por último, pero no por ello menos importante, un agradecimiento especial a mi familia y amigos por su apoyo incondicional durante toda la carrera.

DESARROLLO DE UNA NUEVA FUNCIONALIDAD INTEGRADA EN LA APLICACIÓN PATIO PRO DE 4TPM

Autor: Nadal, Jean.

Director: Thierry, Eric.

Entidad Colaboradora: 4TPM

RESUMEN DEL PROYECTO

El departamento de valores es un servicio ofrecido por bancos y demás instituciones financieras que se encarga de la gestión de todas las operaciones relativas a la negociación de un activo financiero. Se trata de un entorno en permanente evolución debido particularmente al aumento del uso de la informática. Esta transformación se caracteriza también por la aparición de nuevos instrumentos financieros cada vez más sofisticados y a un aumento de la gama de servicios que ofrece este departamento lo que requiere de un uso cada vez más importante de herramientas informáticas.

Debido a esta digitalización y a la creciente complejidad, el sector financiero demanda cada vez más ingenieros y pide a sus trabajadores que dispongan de nociones de programación y de informática en general para entender sus estrategias y programas. Es por lo tanto cada vez más habitual cruzarse con un gran número de ingenieros que desempeñan diferentes empleos en empresas del sector financiero.

En este entorno se incluye este proyecto de fin de carrera, nacido a partir de unas prácticas de una duración de 5 meses realizadas en 4TPM, empresa francesa de software para trading y gestión de carteras. Son los desarrolladores de la plataforma PATIO utilizada por bancos y sociedades de gestión de carteras y el proyecto se realiza en su equipo de desarrollo.

Una de las funcionalidades que permite PATIO a sus usuarios es la de ejecutar órdenes a mercado en bloque. Esto permite a un gestor ejecutar una sola orden en la que se realizan varias acciones para varios de sus clientes. Este tipo de ordenes genera una tarea compleja que PATIO trata de forma asíncrona. Las tareas llegan a unos servidores que en función de su complejidad van a dividirla, o no, en un número variable de subtareas que van a asignar posteriormente a otros servidores para ejecutarlas. Se trata de un proceso complejo lo que dificulta, en el caso de que haya un problema, saber en qué momento y en que servidor se ha producido el fallo.

Hasta ahora, cuando un cliente tiene un problema, el equipo de soporte busca manualmente el momento y servidor en el que se ha producido. Para eso tienen que buscar en los denominados logs de las tareas asíncronas. Los logs son unos ficheros que contienen el registro de todos los acontecimientos de un proceso. Esto permite analizar paso a paso la actividad interna del proceso al igual que sus interacciones con su entorno. Debido a su complejidad esta búsqueda requiere tiempo y no siempre es posible en particular cuando los servidores se hallan en las oficinas de los clientes y que no se tiene acceso directo a los logs.

Hoy en día, las empresas buscan aumentar su efectividad mediante, por ejemplo, el aumento de la agilidad y rapidez de sus procesos internos y de los servicios que ofrecen. Una forma de aumentar dicha efectividad es mediante la automatización informática de procesos

que anteriormente se realizaban a mano. Esto permite una reducción de tiempo considerable debido a la inmensa capacidad de cálculo que tienen hoy en día los ordenadores.

La motivación principal para llevar a cabo este proyecto es por lo tanto la de automatizar un proceso interno que es largo, tedioso y no siempre posible. Esto va a permitir reducir el proceso de búsqueda de los logs a cero y convertirlo en algo inmediato. De este modo se podrá encontrar rápidamente el fallo y resolver el problema del cliente que es al fin y al cabo lo más importante en este tipo de situaciones.

La estructura del proyecto es de tipo cliente – servidor y se divide en dos funciones diferentes presentes en ambas partes. La primera función, Display, permite obtener toda la información relativa a las tareas, subtareas y servidores asíncronos. La función Monitor se usa para ver de forma inmediata el estado de los servidores al igual que de las tareas y subtareas.

A continuación se muestra un esquema en el que se entiende la relación que hay entre las diferentes partes que forman la plataforma PATIO, sus conexiones con el entorno y en qué partes se integra el proyecto.

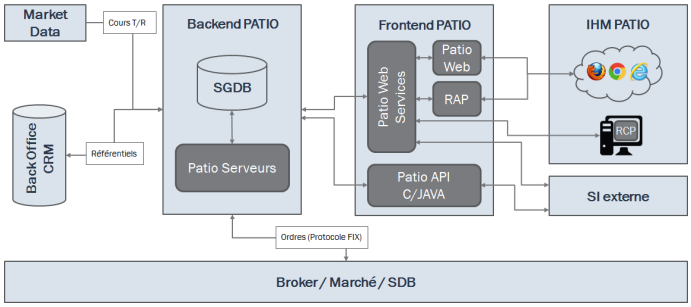


Figura 1: Arquitectura de la plataforma PATIO

La parte BACKEND representa la parte servidor en la cual se han añadido y modificado ciertos archivos en C que se encargan de realizar las consultas SQL a las tablas de servidores, tareas y subtareas.

Por otro lado la parte FRONTEND es la parte cliente donde se han añadido un número importante de clases. Esto se debe a que en esta parte ha sido necesaria la creación de toda la estructura al ser, en el contexto de este proyecto, la primera vez que se han manipulado datos relativos a servidores, tareas o subtareas asíncronas en los servicios web de la empresa. Esta parte, escrita en JAVA, permite la realización de la interfaz con la que el usuario va a realizar las consultas que desee al igual que se encarga de la comunicación con el servidor.

El proyecto empezó con el desarrollo de la función Display que, al ser más complicada y completa, permite entender desde el comienzo el funcionamiento de la comunicación de la estructura cliente – servidor de la empresa.

Se desarrolla en primer lugar la parte servidor, encargada de crear la consulta a la base de datos con la información que le llega del cliente. Esta parte se escribe en lenguaje C y siguiendo un planteamiento bottom up es decir empezando por abajo, en este caso la base de datos, para llegar finalmente a la comunicación con el cliente. Hay que añadir ciertas funciones y estructuras a los archivos ya existentes que gestionan las acciones relativas a las tareas, async_task.c, las subtareas, async_subtask.c, los servidores, async_registration.c y sus

respectivos archivos de cabecera. En cada caso las modificaciones son muy parecidas y van orientadas a añadir la opción de realizar consultas a su tabla en la base de datos respectiva.

En los archivos de cabecera se añade una estructura con el objetivo de almacenar la información que llegue desde el cliente y poder crear la consulta en base a dicha información. En esta estructura se definen las mismas variables que tiene la tabla a la que se hará la consulta, de forma que se podrán hacer consultas filtrando con cualquier variable de la tabla.

En los archivos .c se añade en primer lugar un tipo de estructura que va a permitir la codificación de las variables definidas anteriormente, a la hora de enviar la información al cliente, y la descodificación de dicha información cuando se reciba. Este procedimiento ya existía y se realiza en todos los archivos del servidor de la empresa. A continuación se añaden las 3 funciones siguientes en cada uno de los archivos. La primera es async_nombre_de_la_tabla_from_objetrequest donde, en función del tipo de consulta, que en nuestro caso es de tipo lectura, asocia el código mess_tag y el ID de la tabla y realiza la llamada a la segunda función, async_nombre_de_la_tabla_server_process. Esta función se encarga de identificar el tipo de consulta y de llamar a la función encargada de realizar la acción. En este caso son consultas de lectura por lo que se hace la llamada a la tercera función, async_nombre_de_la_tabla_server_read. Se trata de la función más importante puesto que es en ella donde se analiza la consulta del cliente, se crea la consulta SQL a la base de datos, se envía, se almacena el resultado de dicha consulta y finalmente se envía de vuelta al cliente la información solicitada.

A continuación se muestra la consulta SQL que se crea en el caso de que la información recibida proveniente del cliente sea el estado y la fecha de creación de una tarea asíncrona:

```
SELECT *
FROM ASYNC_TASK
WHERE DTCREAL = “fecha de creación” AND STATUSI = “estado”
```

Esta es la dinámica general en la que se basa el servidor para esta función, sin embargo hay ciertas diferencias particularmente en la función server_read y en la manera en que se crean las consultas SQL entre los archivos async_task, async_subtask y async_registration. Esto se debe simplemente a que la información que contienen sus respectivas tablas es diferente.

Una vez el servidor realizado se puede empezar la parte cliente. Para esta parte ha sido necesaria la creación de nuevos archivos. Esto se debe a que hasta el momento, no había ningún tránsito de información sobre los servidores, tareas o subtareas entre el servidor y los servicios web por lo que es necesario crear toda la estructura en la parte cliente.

Lo primero que hay que crear son las APIJAVA, que se encargan de la comunicación con el servidor. Se requiere de la creación de 6 clases, una para cada tipo de dato (servidor, tarea y subtarea) y su respectiva clase que gestiona la consulta. En el primer grupo se definen las variables que componen su respectiva tabla. En las clases de consulta se definen las variables con las que se puede filtrar, en este caso son las mismas que las que componen la tabla de modo que potencialmente se puede filtrar gracias a cualquiera de las variables de su tabla. Además de esto se añaden dos funciones, la primera construirá la consulta con los parámetros introducidos por el usuario. La segunda función crea la socket que une al servidor, realiza la llamada a la función anterior, y recibe el resultado del servidor.

Las capas superiores sin ser más complejas de programar, si requieren de un número muy importante de clases. Estos se debe a que en muchos casos es necesaria la realización de una clase para cada tipo de objeto, obteniendo de esta forma tres clases muy similares que realizan la misma acción en el caso de las tareas, subtareas y servidores. A continuación se explica la utilidad de dichas clases centrándose únicamente en aquellas que se encargan de lo relativo a las tareas asíncronas.

Lo primero que se debe hacer es crear un objeto en el que se definen las variables de la tabla de tareas, no es obligatorio que tengan el mismo nombre que en las APIJAVA puesto que es necesaria la creación de una clase que haga de adaptador asociando el contenido de las variables de tipo APIJAVA con el de las variables de esta clase. En este caso también se crea una clase de tipo consulta en la que únicamente se definen las variables con las que el usuario puede filtrar su búsqueda. En el caso de las tareas se ha escogido el estado, el usuario que creó la tarea, la fecha de creación y las horas de inicio y de fin.

A continuación se crean un grupo de clases que permiten la apertura de la funcionalidad en la plataforma, un editor para la ventana de parámetros y otro para la de resultados. La primera clase es muy básica y simplemente se especifica el nombre del editor y el de la función. La segunda es de las más importantes puesto que hay que escribir toda la parte gráfica con la que interactúa el usuario para escoger los parámetros de búsqueda. Además es en esta clase donde se crea la consulta con los parámetros que acaba de introducir el usuario y se invoca al servicio que la transmite a las APIJAVA. La última función es también un editor que en este caso se encarga de la apariencia gráfica del resultado. En ella se utilizan dos clases adicionales, la primera permite exportar la tabla de resultados a varios formatos, PDF por ejemplo. Esta clase se hace en XML aunque hay una interfaz en Eclipse que permite crearla manualmente. La segunda se encarga de la definición de las columnas de la tabla de resultados y de rellenar cada una de las celdas con el resultado adecuado.

Una vez que se han creado estas clases hay que juntarlas mediante extensiones para que sean visibles en la plataforma. Para ello existe un plug-in que permite la definición de las extensiones, también se puede hacer escribiéndolo directamente usando XML lo que en ocasiones resulta más sencillo. En un primer tiempo se especifican las clases encargadas de la apertura y de la edición de la ventana de parámetros y de resultados de la función. A continuación se especifica en qué condiciones se quiere que la función sea visible y esté disponible. En este caso se desea que la función sea visible cuando nos encontremos en la categoría Async Monitor y cuando la subcategoría seleccionada sea la tarea. La función debe estar disponible cuando una de las categorías esté seleccionada.

Estas son las principales clases que se han tenido que crear, no obstante también se han escrito ciertas clases necesarias para el funcionamiento de cualquier funcionalidad pero que no tienen especial interés puesto que se trata de clases cortas y genéricas.

Llegados a este punto la función Display está terminada y operativa siguiendo las especificación iniciales y se puede empezar la programación de la función Monitor.

A pesar de tener que crear todos los archivos para esta función, incluido el servidor, su desarrollo es muy similar al de la función Display, puesto que realiza las consultas a las mismas tablas de la base de datos. Tanto en el servidor como en la parte cliente se han creado los mismos archivos y clases y se ha seguido el mismo procedimiento.

Una vez que se ha concluido el desarrollo de la función Monitor y por ende el del proyecto tal y como se imaginó en un principio, se añaden ciertas mejoras orientadas a facilitar la búsqueda de información por parte del equipo de soporte. En el caso de la búsqueda de servidores y de subtareas se añaden ciertos campos en la ventana de parámetros. En el caso de las tareas lo que se añade es una vista en detalle disponible en la ventana de resultados. Se abre al hacer doble clic sobre cualquiera de las tareas y en ella se despliegan las subtareas en las que se divide la tarea seleccionada.

A continuación se muestra el resultado final, incluyendo las mejoras, para el caso de una búsqueda de tareas, en ella el usuario ha introducido una fecha cualquiera:

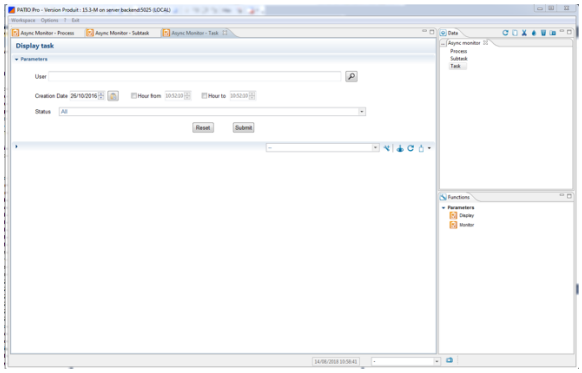


Figura 2: Ventana de parámetros de la función Display para una tarea

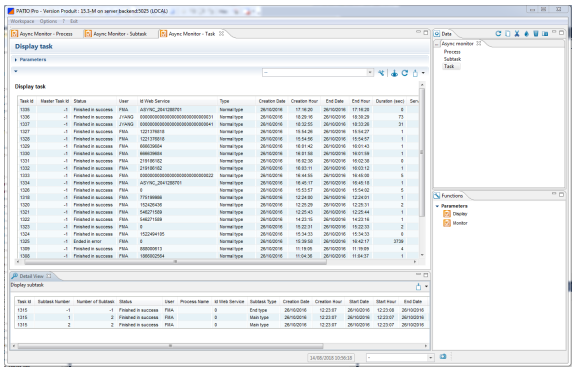


Figura 3: Resultado con vista en detalle de la búsqueda de una tarea

Con la aparición de las nuevas tecnologías nos hemos acostumbrado a recibir la información de forma instantánea, vivimos en la era de la inmediatez. Este concepto se aplica a todos los aspectos de la vida, tanto en el mundo laboral como en el día a día.

Ser capaces de ofrecer un servicio de forma rápida y eficaz se ha convertido por lo tanto en un desafío fundamental que todas las empresas deben afrontar. Es aún más cierto en el mercado financiero y en particular en un servicio de ejecución de órdenes en tiempo real.

Esta es la idea principal en la que se basa este proyecto, que busca dotar a la empresa de una herramienta que le permita ser más rápida, eficaz y reactiva.

Una vez se ha concluido el proyecto se puede afirmar que el objetivo se ha cumplido. Gracias a las dos funciones que se han realizado se puede proporcionar al cliente una respuesta mucho más rápida sobre porque ha fallado su operación y a raíz de esto ofrecerle una solución en un plazo más corto. También permite llevar un seguimiento más preciso del estado de los servidores para poder corregir los fallos con mayor brevedad incluso antes de que el cliente tenga que contactar con la empresa, de forma que cuando lo haga ya se le pueda proporcionar una solución de forma inmediata. De este modo se le ofrece al cliente un servicio de mayor calidad, lo más rápido y eficaz posible.

DEVELOPMENT OF A NEW FUNCTIONALITY INTEGRATED IN THE PATIO PRO APPLICATION OF 4TPM

Author: Nadal, Jean.

Director: Thierry, Eric.

Collaborating Institution: 4TPM

PROJECT SUMMARY

The securities department is a service offered by banks and other financial institutions that is responsible for the management of all transactions related to the trading of a financial asset. This is a constantly changing environment, particularly due to the increased use of information technology. This transformation is also characterized by the appearance of new sophisticated financial instruments and new services offered by this department, which requires an increasingly important use of computer tools.

Due to this digitization and increasing complexity, the financial sector is demanding more and more engineers and requires its employees to have programming and IT skills in general to understand its strategies and programs. It is therefore common to meet a large number of engineers who work in different jobs in the financial sector.

This environment includes this end-of-degree project, born from a 5-month internship at 4TPM, a French software company for trading and portfolio management. They are the developers of the PATIO platform used by banks and portfolio management companies and the project is carried out in their development team.

One of the functionalities that allows PATIO to its users is that of executing market bloc orders. This allows a manager to execute a single order in which several actions are performed for several of its clients. This type of order generates a complex task that PATIO handles asynchronously. The tasks arrive at some servers that, depending on their complexity, will divide them, or not, into a variable number of subtasks that they will assign later to other servers to execute them. This is a complex process which makes it difficult, in the event of a problem, to know when and on what server the failure occurred.

Until now, when a client has a problem, the support team manually searches for the time and server it has occurred. For that they have to search in the so-called logs of asynchronous tasks. Logs are files that contain the log of all the events of a process. This allows you to analyze step by step the internal activity of the process as well as its interactions with its environment. Due to its complexity, this search is time-consuming and not always possible, particularly when servers are located in the client's offices and there is no direct access to the logs.

Today, companies seek to increase their effectiveness by, for example, increasing the agility and speed of their internal processes and the services they offer. One way to increase this effectiveness is by computerizing processes that were previously done by hand. This allows for a considerable reduction in time due to the immense computing power of computers today.

The main motivation for carrying out this project is therefore to automate an internal process that is long, tedious and not always possible. This will allow you to reduce the process

of searching for the logs to zero and make it immediate. This way you can quickly find the fault and solve the customer's problem, which is, after all, the most important thing in this type of situation.

The structure of the project is client-server and is divided into two different functions present on both sides. The first function, Display, allows you to obtain all the information related to tasks, subtasks and asynchronous servers. The Monitor function is used to immediately view the status of servers as well as tasks and subtasks.

Below is a diagram showing the relationship between the different parts that make up the PATIO platform, their connections with the environment and the parts in which the project is integrated.

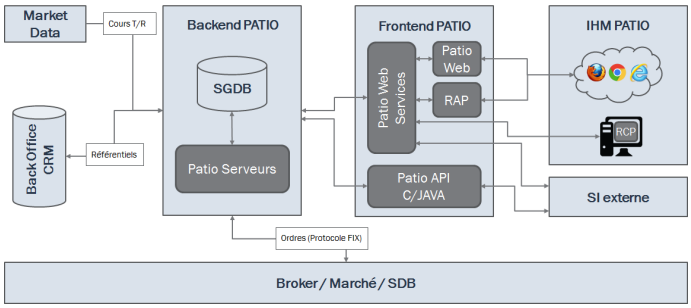


Figure 1: Architecture of the PATIO platform

The BACKEND part represents the server part in which certain C files have been added and modified to perform SQL queries to the server tables, tasks and subtasks.

On the other hand, the FRONTEND part is the client part where an important number of classes have been added. This is due to the fact that in this part it has been necessary to create the whole structure as it is, in the context of this project, the first time that data related to servers, tasks or asynchronous sub-tasks have been manipulated in the company's web services. This part, written in JAVA, allows the creation of the interface with which the user can make the queries he wishes, as well as the communication with the server.

The project began with the development of the Display function, which, being more complicated and complete, allows us to understand from the outset how the communication of the company's client-server structure works.

First of all, the server part is developed, in charge of creating the query to the database with the information received from the client. This part is written in C language and following a bottom-up approach, i.e. starting at the bottom, in this case the database, and finally reaching the communication with the customer. Certain functions and structures need to be added to existing files that manage actions related to tasks, async_task.c, subtasks, async_subtask.c, servers, async_registration.c and their respective header files. In each case the modifications are very similar and are aimed at adding the option of making queries to its table in the respective database.

In the header files a structure is added in order to store the information coming from the client and create the query based on this information. This structure defines the same variables

as the table to which the query will be made, so that queries can be made by filtering with any variable of the table.

In the .c files a type of structure is first added that will allow the coding of the variables defined above, when sending the information to the client, and the decoding of this information when it is received. This procedure already existed and is performed on all files on the company's server. The following 3 functions are then added to each of the files. In the first one, depending on the type of query, which in our case is read type, it associates the mess_tag code and the ID of the table and makes the call to the second function. This function is responsible for identifying the type of query and calling the function responsible for carrying out the action. In this case they are read queries so the third function, is called. This is the most important function since it is where the client's query is analyzed, the SQL query is created to the database, the result of the query is sent, stored and finally the information requested is sent back to the client.

Below is the SQL query that is created if the information received from the client is the status and date of creation of an asynchronous task:

```
SELECT *
FROM ASYNC_TASK
WHERE DTCREAL = “date of creation” AND STATUSI = “status”
```

This is the general dynamic on which the server is based for this function, however there are certain differences particularly in the server_read function and in the way the SQL queries are created between the async_task, async_subtask and async_registration files. This is simply because the information contained in their respective tables is different.

Once the server is done you can start the client part. For this part it has been necessary to create new files. This is because until now, there was no transit of information about servers, tasks or subtasks between the server and web services so it is necessary to create the entire structure on the client side.

The first thing to create are the APIJAVAs, which are responsible for communication with the server. It requires the creation of 6 classes, one for each type of data (server, task and subtask) and its respective class that manages the query. The first group defines the variables that make up its respective table. In the query classes, the variables with which you can filter are defined, in this case they are the same as those that make up the table so that they can potentially be filtered thanks to any of the variables in your table. In addition to this two functions are added, the first one will construct the query with the parameters entered by the user. The second function creates the socket that connects to the server, makes the call to the previous function, and receives the result from the server.

The upper layers without being more complex to program, require a very important number of classes. Because in many cases it is necessary to perform a class for each type of object, thus obtaining three very similar classes that perform the same action in the case of tasks, subtasks and servers. The usefulness of these classes is explained below, focusing only on those that deal with asynchronous tasks.

The first thing to do is to create an object in which the variables of the task table are defined, it is not mandatory that they have the same name as in the APIJAVA since it is

necessary to create a class that acts as an adapter associating the content of the APIJAVA variables with that of the variables of this class. In this case, a type of query type is also created in which only the variables with which the user can filter his search are defined. In the case of tasks, the status, the user who created the task, the creation date and the start and end times have been chosen.

Next, a group of classes is created that allow the opening of the functionality in the platform, an editor for the parameters window and another for the results window. The first class is very basic and simply specifies the name of the editor and the function. The second one is one of the most important since you have to write the whole graphical part with which the user interacts to choose the search parameters. It is also in this class where the query is created with the parameters just entered by the user and the service is invoked to transmit it to the APIJAVA. The last function is also an editor that in this case is in charge of the graphic appearance of the result. Two additional classes are used, the first one allows exporting the results table to various formats, PDF for example. This class is done in XML although there is an interface in Eclipse that allows you to create it manually. The second is responsible for defining the columns of the results table and filling each of the cells with the appropriate result.

Once these classes have been created, they must be joined together using extensions to make them visible on the platform. To do this there is a plug-in that allows the definition of the extensions, you can also do it directly using XML which is sometimes easier. The first step is to specify the classes responsible for opening and editing the parameters and results window of the function. You then specify under which conditions you want the function to be visible and available. In this case you want the function to be visible when you are in the Async Monitor category and when the selected subcategory is the task. The function must be available when one of the categories is selected.

These are the main classes that have had to be created, however, certain classes have also been written that are necessary for the operation of any functionality but that are not of special interest since they are short and generic classes.

At this point the Display function is finished and operational following the initial specifications and you can start programming the Monitor function.

Despite having to create all the files for this function, including the server, its development is very similar to that of the Display function, since it queries the same tables in the database. The same files and classes have been created on the server as on the client side and the same procedure has been followed.

Once the development of the Monitor function has been completed, and therefore of the project as originally imagined, certain improvements are added to facilitate the support team's search for information. In the case of server and subtask searches, certain fields are added in the parameter window. In the case of tasks, what is added is a detail view available in the results window. It opens when you double-click on any of the tasks and displays the subtasks into which the selected task is divided.

Below is the final result, including the improvements, for the case of a task search, in which the user has entered any date:

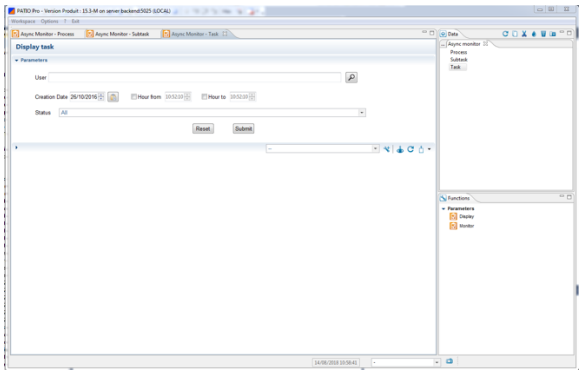


Figure 2: Parameter window of the Display function for a task

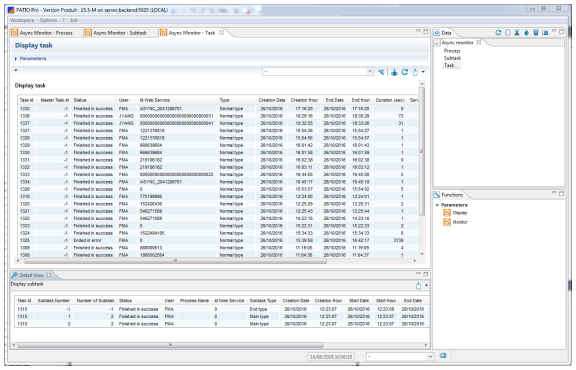


Figure 3: Result with detailed view of the search for a task

With the advent of new technologies we have become accustomed to receiving information instantly, we live in the era of immediacy. This concept applies to all aspects of life, both in the workplace and in everyday life.

Being able to offer a service quickly and efficiently has therefore become a fundamental challenge that all companies must face. This is even more true in the financial market and in particular in a real-time order execution service.

This is the main idea behind this project, which seeks to provide the company with a tool that allows it to be faster, more efficient and more reactive.

Once the project has been completed, it can be said that the objective has been achieved. Thanks to the two functions that have been performed, it is possible to provide the customer with a much faster response as to why his operation has failed and, as a result, to offer him a solution in a shorter period of time. It also allows for more accurate tracking of server status so that faults can be corrected more quickly even before the customer has to contact the company, so that when they do, a solution can be provided immediately. In this way, the customer is offered the highest quality service, as quickly and efficiently as possible.

Índice

Índice de figuras

1. Introducción

1.1 Contexto

El departamento de valores es un servicio ofrecido por bancos y demás instituciones financieras que consiste en la custodia y ejecución de las órdenes de compraventa y suscripción de valores por cuenta de sus clientes. Se encargan por lo tanto de la gestión de todas las operaciones posteriores a la negociación relacionadas con sus carteras de valores. Se trata de la vida de un activo financiero una vez que ha sido comprado por un actor y vendido por otro y concierne a todas las clases de instrumentos financieros.

Se trata de un entorno en permanente evolución de un punto de vista estructural y técnico debido al aumento del uso de la informática en el sector financiero, pero también reglamentario con la aparición de nuevas directivas como lo son MiFID II, directiva europea relativa a los mercados e instrumentos financieros que completando la MiFID busca garantizar la transparencia de los mercados e incrementar la protección a los inversores particularmente los minoristas, y la nueva ley de protección de datos. Esta transformación se ha caracterizado también por la aparición de nuevos instrumentos financieros cada vez más sofisticados y a un aumento de la gama de servicios que ofrece este departamento. Hoy en día la mayoría de las empresas no se encargan únicamente de la tarea de custodio sino de un extenso número de actividades que requieren de un uso cada vez más importante y sofisticado de herramientas informáticas.

Debido a esta digitalización y a la creciente complejidad, el sector financiero demanda cada vez más ingenieros y pide a sus trabajadores que dispongan de nociones de programación, para que sean capaces de realizar ellos mismos sus modelos, y de informática en general para ayudar a la hora de entender como sus estrategias y programas interactúan con los servidores. Es por lo tanto cada vez más habitual cruzarse con un gran número de ingenieros que desempeñan diferentes empleos en empresas del sector financiero.

En este entorno se incluye este proyecto de fin de carrera, nacido a partir de unas prácticas de una duración de 5 meses realizadas en la empresa 4TPM. Se trata de una empresa francesa de software para trading y gestión de carteras. Son los desarrolladores de la plataforma PATIO utilizada por bancos y sociedades de gestión de carteras.

Dichas prácticas forman parte del diploma realizado en la Ecole Centrale de Nantes, son las denominadas prácticas de ingeniero y requieren por lo tanto de dicho nivel. Son obligatorias para completar el diploma en Francia y representan una etapa importante en la carrera de cualquier estudiante de ingeniería al ser, en la mayoría de los casos, el contacto con el mundo laboral de mayor duración e interés de un punto de vista de la ingeniería.

El proyecto se realizará en el equipo de desarrollo de la empresa, que se dedica a implementar soluciones informáticas para resolver problemas, añadir mejoras o nuevas funcionalidades que van dirigidas tanto a los clientes como a la propia firma. Siendo su tarea principal la de desarrollar una nueva funcionalidad para el equipo de soporte que permitirá la automatización de un proceso interno de la empresa. Será por lo tanto de uso exclusivamente interno y en ningún caso se implementará en la plataforma de los clientes.

Una de las funcionalidades que permite PATIO a sus usuarios es la de ejecutar órdenes a mercado en bloque. Es decir, si un gestor tiene una cartera con un perfil determinado de clientes y quiere realizar, por ejemplo, la compra de un activo para todos sus clientes con dicho perfil, tiene la posibilidad de ejecutar una sola orden conjunta que permita a todos los clientes seleccionados la compra de dicho activo sin tener la necesidad de realizarla con todos los clientes uno a uno.

Este tipo de órdenes generan una tarea compleja que PATIO trata de forma asíncrona. Las tareas llegan a unos servidores denominados de cabeza que en función de su complejidad van a dividirla, o no, en un número variable de subtareas que van a asignar posteriormente a otros servidores que van a ejecutarlas. Se trata de un proceso complejo lo que dificulta, en el caso de que haya un problema, saber en qué momento y en que servidor se ha producido el fallo.

A día de hoy cuando un cliente tiene un problema, el equipo de soporte busca manualmente el momento y servidor en el que se ha producido. Para eso tienen que buscar los denominados logs de las tareas asíncronas. Los logs son unos ficheros que contienen el registro de todos los acontecimientos de un proceso. Esto permite analizar paso a paso la actividad interna del proceso al igual que las interacciones con su entorno. Debido a su complejidad esta búsqueda requiere tiempo y no siempre es posible.

Se busca por lo tanto ofrecer una solución a este problema.

1.2 Motivación

Hoy en día, las empresas buscan aumentar su efectividad mediante, por ejemplo, el aumento de la agilidad y rapidez de sus procesos internos y de los servicios que ofrecen. Una forma de aumentar dicha efectividad es mediante la automatización informática de procesos que anteriormente se realizaban a mano. Esto permite una reducción de tiempo impresionante debido a la inmensa capacidad de cálculo que tienen hoy en día los ordenadores.

La motivación principal para llevar a cabo este proyecto es por lo tanto la de automatizar un proceso interno que es largo, tedioso y no siempre posible. Esto va a permitir reducir el proceso de búsqueda de los logs a cero y convertirlo en algo inmediato. De este modo se podrá encontrar rápidamente el fallo y resolver el problema del cliente que es al fin y al cabo lo más importante en este tipo de situaciones.

Resulta especialmente interesante la posibilidad de desarrollar un proyecto de forma integral en el seno de una empresa. Desde la comprensión del problema y el análisis de los objetivos y especificaciones iniciales, hasta la implementación final del producto en un entorno real. Es una oportunidad extraordinaria de descubrir cómo se realiza realmente un proyecto en una empresa.

Personalmente, me parece particularmente atractivo el hecho de realizar un proyecto que satisface una necesidad real de una empresa. No se trata de llevar a cabo un trabajo con

objetivo puramente académico sino de buscar constantemente la utilidad de lo que se esté haciendo para que sea lo más eficaz posible. Es interesante para la formación de un ingeniero que tenga la oportunidad en algún momento de su carrera de poner en práctica sus conocimientos en un proyecto que cumpla este tipo de necesidades.

1.3 Objetivos

El objetivo principal es el de proporcionar a través de una interfaz, al equipo de soporte, una solución que les permita encontrar rápidamente el nombre de los logs de las tareas asíncronas que han fallado. Para posteriormente recuperar dichos logs, analizarlos y poder resolver el error.

Una vez implementada la nueva funcionalidad, esta debería permitir también que la firma sea más reactiva, y se anticipe a la posible llamada del cliente una vez que este detecte que ha habido un fallo cuando intentaba ejecutar una orden. Se podrá consultar de forma habitual el estado de las tareas y servidores y en el momento en el que se detecte un fallo empezar a buscar una solución. De este modo se gana en efectividad y se ofrece al cliente un mejor servicio.

Gracias a esta nueva funcionalidad se podrá llevar un seguimiento más preciso de las tareas que se están llevando a cabo en tiempo real al igual que el estado en el que se encuentran los servidores que las están ejecutando. Un dato que se proporcionará al usuario es el del last heartbeat, se trata de la fecha y hora del último momento en que el servidor comunicó que estaba funcionando. También podrá realizar una búsqueda introduciendo un valor para esta variable para que solo aparezcan los servidores que llevan más de X tiempo sin dar señales de vida. Esto permite detectar antes de tiempo el fallo de un servidor, cuando vemos que este lleva un rato inusualmente largo sin actualizar el valor del heartbeat pero que sin embargo su estado es aparentemente correcto. Este desfase generalmente implica un fallo en el servidor.

La realización de este proyecto me permitirá entender y vivir de primera mano cómo se desarrolla desde el inicio hasta la implementación final, en el entorno real, una nueva funcionalidad, satisfaciendo una necesidad real en el mundo de la empresa.

A lo largo del proyecto se utilizan varios lenguajes de programación, lo que me da la oportunidad de profundizar e incluso de descubrir lenguajes que no había estudiado hasta el momento, como son JAVA, SQL y XML. Además, pondré en práctica en un entorno real y complejo nociones de informática como son las comunicaciones de tipo cliente – servidor, entre diferentes sistemas al igual que los diferentes protocolos que rigen dichas comunicaciones.

A pesar de ser nociones que he estudiado desde un punto de vista teórico varias veces a lo largo de la carrera es particularmente beneficioso poder ponerlos en práctica. Una de las principales diferencias entre el planteamiento académico y el puesto en práctica en una empresa es que, a la hora de buscar una solución a un problema, esta no se hace de forma independiente, sino que desde el inicio se desarrolla integrándola en la estructura existente

en la empresa. Esto implica que, aunque haya muchas soluciones posibles haya que escoger la que más se adapte a la metodología y convenciones propias de la empresa.

2. 4TPM

2.1 Historia

4TPM es una empresa francesa especializada en el negocio de valores desde hace más de 20 años. Se dedican a la creación de softwares especializados en trading y gestión de carteras siendo los desarrolladores de la plataforma PATIO, utilizada por más de 500 gestores diariamente para alcanzar sus objetivos. La historia de la plataforma empezó en el grupo Fininfo, proveedor de información financiera, hasta su compra en 2007 por el grupo suizo SIX que se dedica también a los servicios financieros y a la venta de plataformas de difusión de información financiera internacional en tiempo real. En 2014 la plataforma toma un nuevo camino separándose de SIX Group y forman la empresa 4TPM (4 Trading & Portfolio Management).

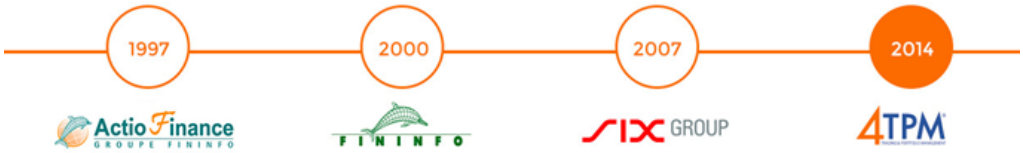


Figura 1: Historia de 4TPM [1]

La empresa la forman 28 colaboradores expertos en mercados financieros y en informática que se dedican en exclusiva a la actividad relativa a la plataforma PATIO y a su labor de desarrollo.

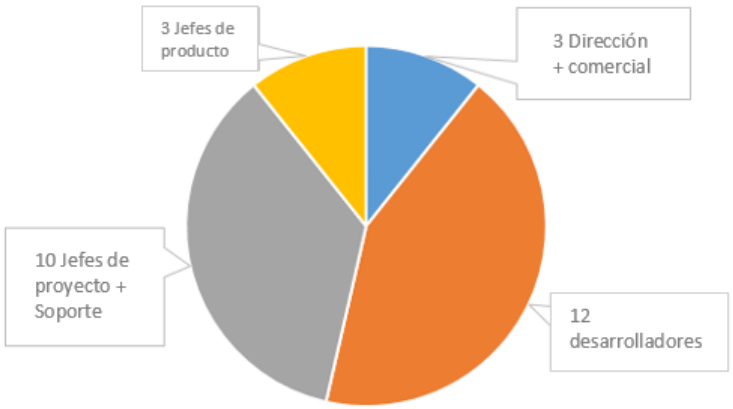


Figura 2: Plantilla de 4TPM[2]

2.2 Tecnología

La plataforma permite un total cumplimiento y gestión del riesgo gracias a un potente motor de reglas que autoriza o no las operaciones de trading en función de parámetros como el perfil del gestor, sus objetivos o de la operación que quiere realizar.

Realizan la transmisión de las órdenes mediante el método STP, con una fiabilidad demostrada por millones de transacciones tratadas con éxito, se trata de una técnica de automatización de la gestión de órdenes financieras sin cortes ni demora. Nacida a raíz de la reforma que reduce a tres días laborables el plazo de liquidación y entrega de las transacciones a partir de su negociación. Se usa este término para referirse a los proyectos llevados a cabo inicialmente por la DTCC, empresa americana de servicios financieros, que unió los sistemas informáticos de los bancos, brókers y gestores al depositario central DTCC con el objetivo de optimizar la velocidad con la cual estas empresas financieras procesan las transacciones de sus clientes con el fin de cumplir con esta nueva normativa.[3]

2.3 Productos

La plataforma PATIO se divide en 3 productos independientes, PATIO PM para la gestión de carteras, PATIO OLT de bolsa online y PATIO OMS que se encarga del sistema de gestión de órdenes.

Figura 3: Productos de 4TPM [4]

No obstante, encontramos ciertas funcionalidades en los tres productos, la gestión de las autorizaciones para las diferentes funcionalidades en función del perfil del usuario, al igual que la oportunidad de crear para cada perfil un sistema de reglas de alertas y bloqueos que PATIO actualiza automáticamente en función del nivel de aprendizaje de la cuenta.

PATIO PM (Portfolio Management): Producto destinado a los gestores de carteras que permite optimizar los activos de sus clientes además de industrializar las tareas de gestión.

PATIO OLT (Online Trading): Dirigida a los clientes de tipo retail que ofrece un servicio completo de bolsa online.

PATIO OMS (Order Management System): Es la herramienta que permite gestionar el circuito que realizarán las transacciones realizadas en bolsa, de forma que se minimicen las intervenciones manuales, asegurando la comunicación entre departamentos, y mediante el método STP.

Las funcionalidades de PATIO están disponibles a través de:

- PATIO Pro: A disposición de usuarios internos mediante:
    - Web / Cliente ligero (RAP) Disponible en intranet
    - Desktop / Cliente pesado (RCP)

- PATIO Web: Pagina web transaccional que se integrará a una página existente.

- PATIO Web Service: Ofrece todas las funcionalidades a través de servicios web

- PATIO API C/Java [5]

2.4 Integración:

Es posible integrar PATIO en cualquier sistema de información existente. 4TPM se encarga de realizar todo o una parte de la integración de sus productos en el entorno del cliente.

A continuación se muestra un esquema en el que se entiende la relación que hay entre las diferentes partes que forman la plataforma PATIO al igual que sus conexiones con el entorno.

La parte BACKEND representa lo que a lo largo del proyecto se considera la parte servidor en la cual se han añadido y modificado ciertos archivos en C.

Por otro lado, la parte FRONTEND es lo que se llama la parte cliente, donde se han añadido un número importante de clases. Esto se debe a que en esta parte ha sido necesaria la creación de toda la estructura al ser, en el contexto de este proyecto, la primera vez que se han manipulado datos relativos a servidores, tareas o subtareas asíncronas en los servicios web de la empresa.

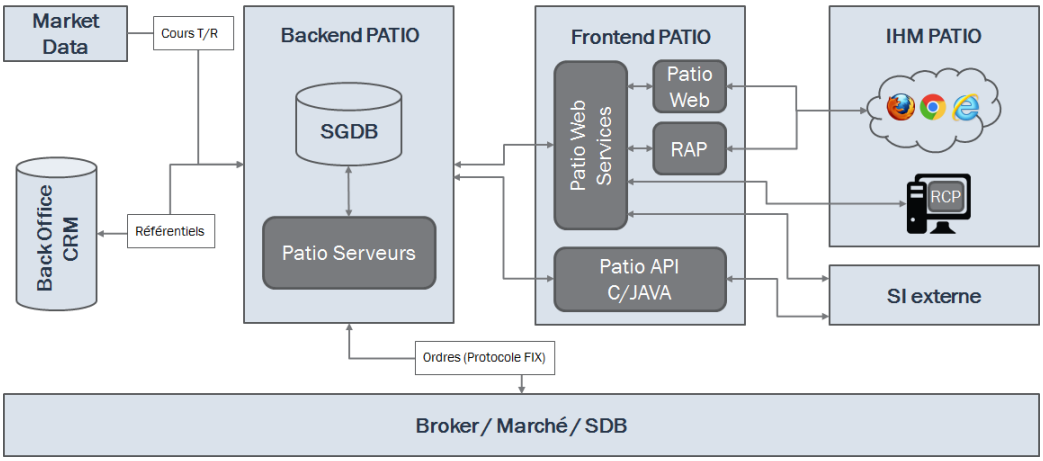


Figura 4: Arquitectura de la plataforma PATIO [6]

2.5 Clientes

Instituciones financieras que usan los productos de 4TPM:

Figura 5: Clientes que usan actualmente PATIO [7]

3. Descripción de las tecnologías. Estado de la técnica

3.1 Estado del proyecto en la empresa

Como se ha dicho anteriormente, el desarrollo del proyecto empieza desde cero, no obstante, hace tiempo que la empresa tiene en mente realizar la automatización del proceso de búsqueda de los logs de las tareas asíncronas que han errado. Por diversas razones, entre las que se incluyen la prioridad que le otorgan a los desarrollos que van dirigidos a los clientes todavía no habían empezado el desarrollo de ninguna solución al problema.

La idea desde el inicio es por lo tanto que el proyecto se encargue, desde su etapa inicial del desarrollo de dicha solución lo que supone el interés añadido de poder observar el desarrollo integral de una funcionalidad.

A pesar de ser una funcionalidad que será exclusivamente de uso interno a la empresa, en particular la usará el equipo de soporte, su uso requiere estar presente en todos los niveles de la estructura de la empresa. Esto se debe a que el usuario realiza una consulta a la base de datos a través de una interfaz implementada en la aplicación PATIO.

Técnicamente para la realización de esta interfaz y su comunicación con el servidor, que realiza las consultas a la base de datos donde se haya el resultado se requieren de varias capas todas realizadas en JAVA. El esquema del proyecto será del tipo cliente – servidor. En este caso la parte desarrollada en JAVA corresponde a la parte cliente. Para permitir realizar la consulta el cliente comunica con la parte servidor escrita en lenguaje C. Dicha parte envía las consultas directamente a las diferentes tablas de la base de datos de la empresa en SQL.

Aunque todavía no existe ninguna solución al proyecto en ningún nivel de la estructura de la empresa en la que apoyarse, las diferentes partes que tiene tanto la parte cliente en JAVA como la parte servidor en C están repletas de archivos que, aunque cumplen funciones muy diferentes a las del proyecto en cuestión, permiten entender cómo se realiza la comunicación entre las bases de datos, servidor y cliente.

Debido a la gran cantidad de programas existentes es posible que se encuentren a lo largo de la realización del proyecto soluciones existentes a puntos particulares. De esta forma es posible inspirarse en ciertos programas, particularmente a la hora de realizar funciones que, para el correcto funcionamiento de un programa, hay que implementar obligatoriamente.

3.2 Procedimiento anterior

El objetivo de utilizar una tarea asíncrona es el de poder dividir la tarea en muchas subtareas que se van a ejecutar cada una en un servidor. El peligro en este caso es que si hay por ejemplo 16 subtareas y una falla, el usuario obtiene un error en la tarea principal pero no la razón ni donde se ha producido. Anteriormente no había forma de saber la naturaleza del fallo ni en que servidor. En la mayoría de los casos al no tener acceso directo de forma externa a PATIO, a estos servidores, particularmente cuando se han desplegado en las oficinas del

cliente. Se tiene que pedir a la empresa todos los logs del día de todos los servidores y se busca de forma manual el error. Esta tarea se convierte por lo tanto en algo parecido a buscar una aguja en un pajar puesto que como se muestra a continuación los logs no son fáciles de leer. Lo que se muestra a continuación es un log real, únicamente uno, los integrantes del equipo de soporte tienen que buscar la información en cientos de ellos.

```
20180502 17:41:51.951 DbExtRead:SQL: SELECT * FROM ASYNC_REGISTRATION WHERE statusi
= 0 AND heartbeat_timestampl < 1508766111 and COMPLEXITYI in (1, 0)
20180502 17:41:51.953 SQL_EXTREAD no more data
20180502 17:41:51.953 async_task_sync_detect_inactive_server : END detect inactive
servers
20180502 17:41:51.953 async_task_sync_task_table : START
20180502 17:41:51.953 async_task_sync_invent_tasks_in_error : START specific( AND
ASYNC_TASK.RESERVEDB = 0)
20180502 17:41:51.953 DbRead:SQL:(nobinding) select ASYNC_TASK.* FROM ASYNC_TASK ,
ASYNC_SUBTASK    where    ASYNC_TASK.ID_TASKL    =    ASYNC_SUBTASK.ID_TASKL    and
ASYNC_TASK.STATUSI = 1 and ASYNC_SUBTASK.STATUSI = 4 AND ASYNC_TASK.RESERVEDB = 0
order by ASYNC_TASK.ID_TASKL ASC
20180502 17:41:51.955 SQL_READ no more data
20180502 17:41:51.955 async_task_sync_invent_tasks_in_error : END found 0
20180502 17:41:51.955 async_task_sync_invent_tasks_finished : START specific( AND
ASYNC_TASK.RESERVEDB = 0)
20180502 17:41:51.955 DbRead:SQL:(nobinding) SELECT * FROM ASYNC_TASK where STATUSI
= 1 AND TASK_TYPEI <> 1  AND ASYNC_TASK.RESERVEDB = 0 and ID_TASKL IN (select distinct
ASYNC_SUBTASK.ID_TASKL   from   ASYNC_SUBTASK   where   statusi   in   (3,   -1)   and
ASYNC_TASK.ID_TASKL    =    ASYNC_SUBTASK.ID_TASKL    minus    select    distinct
ASYNC_SUBTASK.ID_TASKL from ASYNC_SUBTASK where statusi <> 3 and statusi <> -1 and
ASYNC_TASK.ID_TASKL = ASYNC_SUBTASK.ID_TASKL)
20180502 17:41:51.957 async_task_sync_invent_tasks_finished(asynchronous life cycle,
ws = <00000000000000000000000000000072>, id = <852>): Task finished
20180502 17:41:51.957 SQL_READ no more data
20180502 17:41:51.957 async_task_sync_invent_tasks_finished : END found 1
20180502 17:41:52.157 async_task_sync_update_to_error : START Put in error task list
20180502 17:41:52.157 async_task_sync_update_to_error : END Put in error task list
20180502 17:41:52.157 async_task_sync_update_to_finished : START Finish task list
20180502  17:41:52.157  select  *  FROM  ASYNC_SUBTASK  WHERE  ID_TASKL  =  852  AND
SUBTASK_TYPEI = 2
20180502 17:41:52.158 async_task_sync_treatment_epilogue(asynchronous life cycle, ws
= <00000000000000000000000000000072> id = <852>) : START epilogue for DUTUJM :
MESS_TAG_BLOCKPROCESS-BLOCKPROCESS_VALIDATE_BLOCKMASTER)
20180502 17:41:52.158 async_task_sync_treatment_epilogue(asynchronous life cycle, ws
= <00000000000000000000000000000072> id = <852>) : No epilogue treatment specified
20180502 17:41:52.159 async_task_sync_treatment_epilogue(asynchronous life cycle, ws
= <00000000000000000000000000000072> id = <852>) : END epilogue for DUTUJM :
MESS_TAG_BLOCKPROCESS-BLOCKPROCESS_VALIDATE_BLOCKMASTER)
20180502  17:41:52.159  UPDATE  ASYNC_TASK  SET  ID_TASKL=:X1  ,ASSOCIATED_TASKL=:X2
,STATUSI=:X3    ,SUBTASK_NUMBERI=:X4    ,TASK_MESS_TAGI=:X5    ,TASK_TYPEREQUESTI=:X6
,TASK_LOGINS=:X7  ,NOTIFIEDB=:X8  ,DTCREAL=:X9  ,HRCREAL=:X10  ,TIMESTAMP_CREAL=:X11
,DTENDL=:X12 ,HRENDL=:X13 ,TIMESTAMP_ENDL=:X14 ,DURATIONL=:X15 ,WEB_SERVICE_IDS=:X16
,RESERVEDB=:X17 ,TASK_TYPEI=:X18 WHERE  ID_TASKL =  852
20180502 17:41:52.161 Commit  session = 1
20180502 17:41:52.161 async_task_sync_notify_execute(asynchronous life cycle, ws =
<00000000000000000000000000000072>, id = <852>) : START ActiveMQ notification
20180502 17:41:52.162 DEBUG:sqls:SELECT ASYNC_DATAX FROM ASYNC_TASK WHERE id_taskl =
852 :
20180502 17:41:52.163 START reading blob, length:6180:buffer:50000:
20180502 17:41:52.164 END reading BLOB
20180502 17:41:52.164 async_task_sync_update_to_finished : END Finish task list
20180502  17:41:52.164  DbRead:SQL:(nobinding)  SELECT  a.*  FROM  ASYNC_TASK    a,
ASYNC_SUBTASK s where a.STATUSI = 1 and a.TASK_TYPEI = 1 and a.RESERVEDB = 0 and
a.ID_TASKL  =  s.ID_TASKL  and  (s.STATUSI  =  4  or  a.ID_TASKL  IN  (select  distinct
b.ASSOCIATED_TASKL from ASYNC_TASK b where b.STATUSI = 4 and b.TASK_TYPEI = 2 and
a.ID_TASKL  =  b.ASSOCIATED_TASKL  minus  select  distinct  c.ASSOCIATED_TASKL  from
ASYNC_TASK c where c.STATUSI in (1, 0) and c.TASK_TYPEI = 2 and a.ID_TASKL =
c.ASSOCIATED_TASKL))
20180502 17:41:52.166 SQL_READ no more data
20180502 17:41:52.166 async_task_sync_invent_main_associated_tasks_error : END found
0
20180502  17:41:52.166  DbRead:SQL:(nobinding)  SELECT  a.*  FROM  ASYNC_TASK    a,
ASYNC_SUBTASK s where a.STATUSI = 1 and a.TASK_TYPEI = 1 and a.RESERVEDB = 0 and
a.ID_TASKL  =  s.ID_TASKL  and  s.STATUSI  <>  2  and  a.ID_TASKL  IN  (  select  distinct
b.ASSOCIATED_TASKL from ASYNC_TASK b where b.STATUSI = 3 and b.TASK_TYPEI = 2 and
a.ID_TASKL  =  b.ASSOCIATED_TASKL  minus  select  distinct  c.ASSOCIATED_TASKL  from
```

```
ASYNC_TASK c where c.STATUSI in (1, 5, 4, 0) and c.TASK_TYPEI = 2 and a.ID_TASKL =
c.ASSOCIATED_TASKL)
20180502                                                                17:41:52.169
async_task_sync_invent_main_associated_tasks_finished(asynchronous life cycle, ws =
<00000000000000000000000000000072>, id = <851>): Task finished
20180502 17:41:52.169 SQL_READ no more data
20180502  17:41:52.169  async_task_sync_invent_main_associated_tasks_finished  :  END
found 1
20180502 17:41:52.169 async_task_sync_update_to_error : START Put in error task list
20180502 17:41:52.169 async_task_sync_update_to_error : END Put in error task list
```

3.3 Asincronismo

Buscando mejorar la experiencia del usuario, la empresa decidió aumentar el uso del asincronismo en diferentes funcionalidades de la plataforma PATIO, tanto en el caso de los clientes ligeros RAP como pesados RCP. Es particularmente el caso de un usuario que realiza una orden en bloque, que genera una tarea que los servidores trataran de forma asíncrona. Dichas tareas y servidores son los que se estudian a lo largo del proyecto.

Una tarea representa el procesamiento completo de una orden en bloque en este caso, y de cualquier función en general. Siempre se divide en una o varias subtareas a las que se le añaden una subtarea de fin de tratamiento. Esta, permanece inactiva hasta que todas las subtareas hayan terminado su labor. En ese momento se activa y se trata, concluyendo de ese modo la tarea.

Uno de los factores que influyen en el número de subtareas que se van a crear es el número de cuentas que incluye la orden en bloque. Puesto que cada una de las subtareas calcula las órdenes en detalle de un subconjunto de cuentas. La subtarea de final de tratamiento calcula las cantidades de las órdenes globales y actualiza el estado de la orden en bloque. Se puede dar el caso de que una subtarea de fin de tratamiento no tenga ninguna labor que hacer.

El diagrama siguiente muestra los diferentes estados que tienen las tareas y las subtareas.

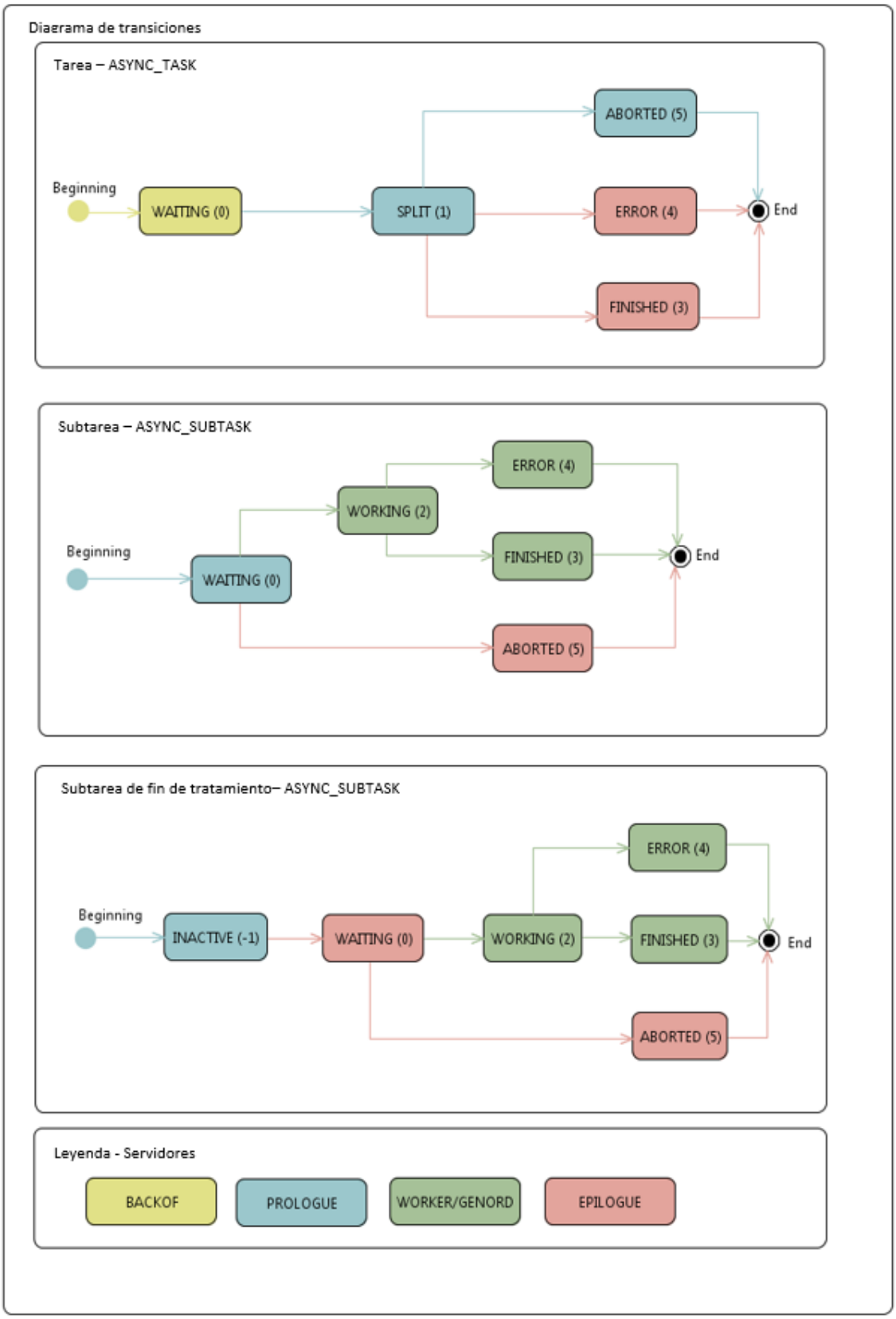


Figura 6: Diagrama de estados de los servidores, tareas y subtareas asíncronas [8]

Con el fin de gestionar el procesamiento de este tipo de tareas la empresa ha desarrollado servidores especializados.

- Servidor de cabeza BACKOF
    - Se asegura de que se puede tratar la tarea y eventualmente impone un bloqueo sobre el dato
    - Incluye la tarea en la tabla de tareas, ASYNC_TASK
    - Anuncia al Front que la tarea ha sido activada, o en su defecto que no puede serlo
- Servidor de reparto PROLOGUE
    - Escanea la tabla de tareas en busca de tareas disponibles y cuando encuentra una en espera:
        - La analiza para determinar su complejidad
        - La descompone en subtareas y las incluye en la tabla de subtareas con el estado “en espera”
        - Añade una tarea de final de tratamiento con el estado “inactiva”
        - Preasigna las subtareas a un servidor de tratamiento en el caso que sea necesario
- Servidor de tratamiento asíncrono WORKER/GENORD
    - Escanea la tabla de subtareas
    - Escoge una subtarea en estado “en espera”
    - Trata la subtarea y actualiza su estado a “acabada” o “en error”
- Servidor de sincronización EPILOGUE
    - Escanea las tablas de tareas y de subtareas
    - Detecta las tareas en error y envía una notificación
    - Detecta los servidores inactivos o en error (en bug) y reasigna sus tareas a servidores que si estén en correcto funcionamiento en el caso que sea necesario
    - Para una tarea cualquiera:
        - En el caso de que todas sus subtareas estén terminadas cambia el estado de la subtarea de final de tratamiento al de “en espera”
        - Si todas las subtareas incluida la de final de tratamiento han terminado, actualiza el estado de la tarea al de “acabada” y envía una notificación

A continuación, se muestra un diagrama que ilustra el funcionamiento de cada servidor:

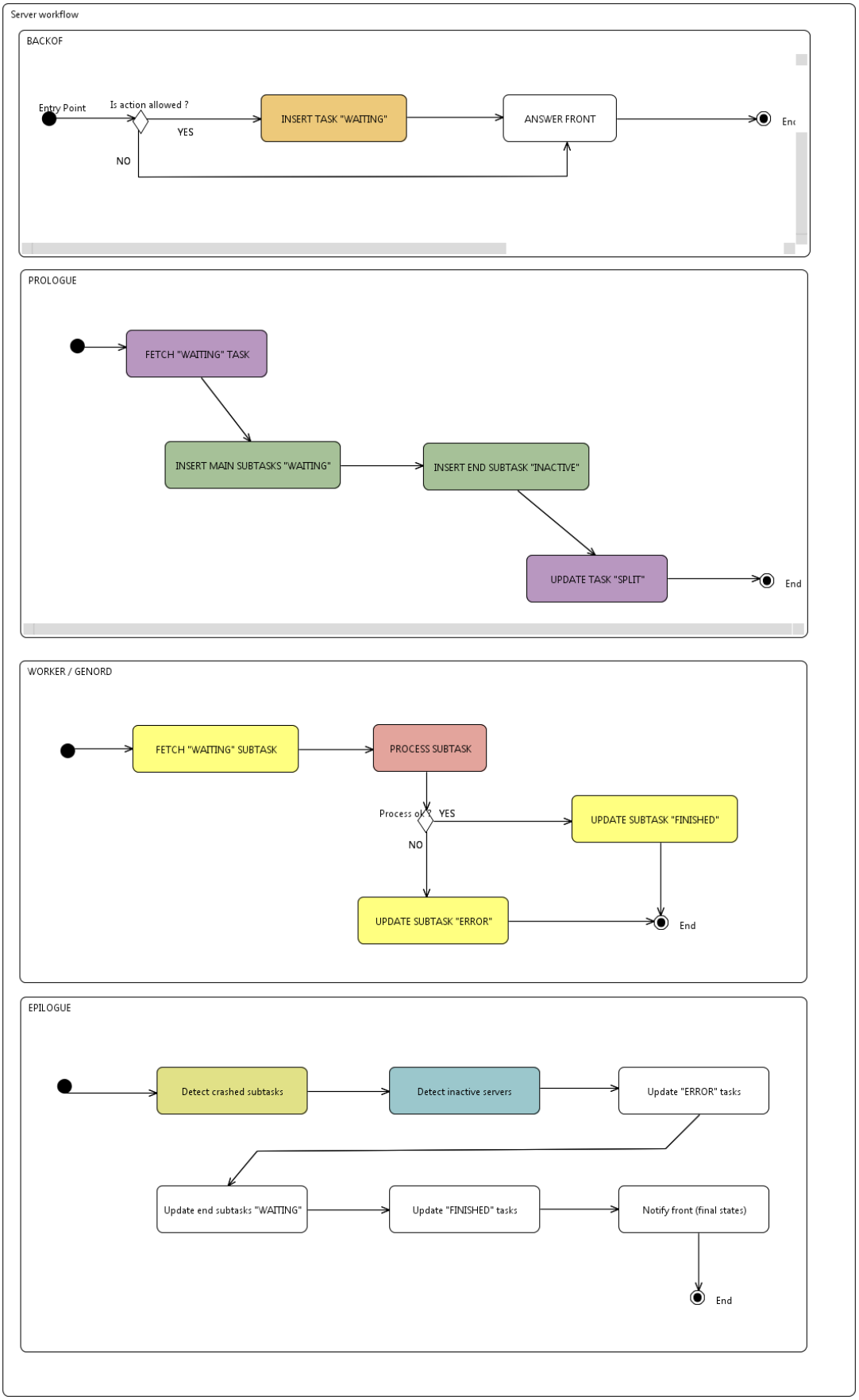


Figura 7: Diagrama del funcionamiento de los diferentes servidores [9]

Siendo este el procedimiento habitual a la hora de tratar las tareas asíncronas se pueden encontrar situaciones particulares en las cuales se actúe de manera diferente como puede ser por ejemplo en el caso de la anulación de una orden en bloque, en cuyo caso se realizarán varias tareas, por lo que se requiere de una tarea maestra que permita vincular dichas tareas.

Anteriormente se ha visto que las tareas, subtareas y servidores se incluían en unas tablas en una base de datos.

La tabla de las tareas, ASYNC_TASK, contiene toda la información relativa a las tareas y permite conocer los datos sobre los que se tiene que ejecutar. A continuación, se muestran las variables disponibles para cada tarea y su significado. [10]

ID_TASKL	Número de identificación de la tarea
ASSOCIATED_TASKL	Identificacion de la tarea maestra para las tareas asociadas
STATUSI	Estado de la tarea
SUBTASK_NUMBERI	Numero de subtareas de la tarea
TASK_MESS_TAGI	MESS_TAGI (código) del tratamiento que la tarea tendrá que ejecutar
TASK_TYPEREQUESTI	TYPEREQUESTI relativo al MESS_TAGI del tratamiento de la tarea
TASK_LOGINS	Login del usuario que ha realizado la acción
NOTIFIEDB	Flag que especifica si el usuario ha sido notificado del fin de la tarea
DTCREAL	Fecha de creación de la tarea
HRCREAL	Hora de creación de la tarea
DTENDL	Fecha de fin de tratamiento de la tarea
HRENDL	Hora de fin de tratamiento de la tarea
DURATIONL	Duración en segundos del tratamiento de la tarea
WEB_SERVICE_IDS	Código de tarea para su búsqueda en los logs
RESERVEDB	Flag para uso en modo standalone como en batch
TASK_TYPEI	Tipo de tarea
ASYNC_DATAX	BLOB: dato que tratar

La tabla de las subtareas, ASYNC_SUBTASK, contiene las informaciones relativas a las subtareas de modo que los servidores asíncronos puedan trabajar sobre ellas si su configuración así lo permite. Contiene también información que permite determinar si el servidor que se supone que está realizando la ejecución de una subtarea lo está haciendo realmente. [11]

ID_TASKL	Número de identificación de la tarea
ID_NUMBERI	Número de la subtarea
SUBTASK_TOTALI	Número total de subtareas de una tarea
DTCREAL	Fecha de creación de la subtarea
HRCREAL	Hora de creación de la subtarea
DTSTARTL	Fecha de comienzo del tratamiento de la subtarea
HRSTARTL	Hora de comienzo de tratamiento de la subtarea
DTENDL	Fecha de fin de tratamiento de la subtarea
HRENDL	Hora de fin de tratamiento de la subtarea
DURATIONL	Duración en segundos del tratamiento de la subtarea
WEB_SERVICE_IDS	Código de la tarea para su búsqueda en los logs
COMPLEXITYI	Tipo de servidor que va a tratar la subtarea
STATUSI	Estado de la subtarea
CLIENT_NAMES	Nombre del servidor que trata la subtarea
SERVER_NAMES	Nombre de la máquina que está tratando la subtarea
SERVER_PORTL	Puerto de la máquina que está tratando la subtarea
RESERVEDB	Flag para uso en modo standalone como en batch
SUBTASK_TYPEI	Tipo de subtarea

ASYNC_REGISTRATION es la tabla que contiene la información sobre los servidores asíncronos. Estos registran y actualizan automáticamente la información que les concierne, los servidores PROLOGUE y EPILOGUE se registran con el fin de poder notificarles en el

momento en el que se necesite su uso. Gracias a esta tabla podemos conocer la prerrogativa que la configuración deja a los servidores, la complejidad de los tratamientos al igual que el uso de la pre asignación para simplificar la búsqueda de tareas. [12]

```
SERVER_NAMES            Nombre de la maquina sobre la que el servidor está funcionando
SERVER_PORTL            Puerto de la maquina sobre la que el servidor está funcionando
CLIENT_NAMES            Nombre del servidor
HEARTBEAT_DATEL         Fecha del último heartbeat (para detectar un servidor en fallo)
HEARTBEAT_HOURL         Hora del último heartbeat (para detectar un servidor en fallo)
STATUSI                 Estado del servidor
COMPLEXITYI             Tipo de servidor
```

Estos servidores se instalan directamente en las oficinas del cliente o en la propia sede de la empresa por un jefe de proyecto, se añaden también ciertas configuraciones particulares en función de las necesidades y funcionalidades de cada cliente.

En el caso de que se halle un problema durante la ejecución de alguna de las funcionalidades, primero se busca en la tabla de las tareas, ASYNC_TASK, para comprobar que la tarea ha sido incluida. En el caso de que no lo haya sido hay que buscar en los logs de los servidores BACKOF. En el caso contrario hay que seguir buscando, ahora en la tabla de las subtareas. En el caso de que no se encuentre, el problema ha ocurrido en los servidores PROLOGUE por lo que hay que buscar en los logs de dicho servidor, si por el contrario la subtarea ha sido incluida correctamente en la tabla es necesario realizar la búsqueda de las subtareas en error. Si se encuentra alguna hay que buscar en los logs del servidor WORKER/GENORD que ha ejecutado la tarea. Todas las búsquedas se realizan mediante consultas SQL directamente a las tablas.

Así se hacía hasta ahora, lo que es largo y no siempre posible ya que en algunos casos los clientes son reacios a que se busque en sus servidores, se niegan a menudo, y en cualquier caso solo es posible hacer consultas particulares, en cambio cuando esté la funcionalidad instalada en PATIO y ya que la plataforma tiene acceso directo se podrá acceder a los datos para buscar los fallos.

4. Proyecto

4.1 Especificaciones iniciales

Para cumplir con los objetivos un jefe de proyecto redacta lo que se denomina como especificaciones y crea una entrada en la herramienta JIRA, donde el encargado de realizar el desarrollo tiene que inscribirse como tal. Se trata de una herramienta en línea cuyo objetivo es el de gestionar las tareas de un proyecto y realizar el seguimiento de bugs e incidentes.

En las especificaciones del proyecto, escritas en francés, tienen que aparecer todas las informaciones que el desarrollador necesita para poder trabajar correctamente. En este caso se puede ver el código de la entrada en JIRA, EVINT-3366, al igual que su título Monitor sobre de tareas asíncronas. A continuación se muestra una traducción del documento original:

EVINT 3366 Monitor sobre las tareas Asíncronas

Objetivo
El objetivo de esta funcionalidad es el de proporcionar al equipo de soporte una solución para encontrar rápidamente el nombre de los log de las tareas asíncronas en error a través del IHM para posteriormente recuperarlos y analizarlos.

Espacio de trabajo: Administración
Categoría: Pilotaje
Dato: Monitor asíncrono

Se mostrarán 3 tipos de datos estáticos:
- Procesos (Process): para la tabla ASYNC_REGISTRATION
- Tareas (Task): para la tabla ASYNC_TASK
- Subtarea (Subtask): para la tabla ASYNC_SUBTASK

Funciones:
- Display
- Monitor

La función Display aparecerá en gris cuando no haya ningún dato seleccionado.
La función Monitor siempre estará disponible.

Monitor:
Esta función se usara para ver rápidamente si hay errores en los procesadores, tareas o subtareas asíncronas.

Esta función mostrará unos contadores sobre las tres tablas. Habrá que prever por lo tanto un objeto por ejemplo ASYNC_MONITOR del lado del servidor que encontrará los contadores sobre las tres tablas ASYNC a través de statusi.

Será necesario que se muestre una tabla (no necesariamente BIRT) que muestre las tres líneas siguientes:

Dato	En error	En curso	En espera	Terminado	Anulado
Proceso	0	15	18		
Tarea	2	5	18	58	0
Subtarea	6	15	0	138	0

Las celdas en error deberán aparecer en rojo si su número es superior a 0.

Display:
En relación al dato Proceso de la tabla ASYNC_REGISTRATION:

Ventana de parámetros:
- Dato: Proceso (estático)
- Estado (Status) Combobox ASYNC_REGISTRATION.statusi
  - ASYNC_DEF_STATE_ZOMBIE 2 “en error”
  - TOUS “Todos”
  - ASYNC_DEF_STATE_WAITING 0 “en espera”

    - ASYNC_DEF_STATE_WORKING 1 “en curso”
- Por defecto ASYNC_DEF_STATE_ZOMBIE

Se mostrará una tabla con la siguiente información:

Columna	Column	Campo
Nombre del host	Host names	SERVER_NAMES
Puerto	Port	SERVER_PORTL
Nombre del proceso	Process name	CLIENT_NAMES
Último heartbeat (DT)	Last heartbeat (DT)	HEARTBEAT_DATEL
Último heartbeat (HR)	Last heartbeat (HR)	HEARTBEAT_HOURL
Estado	Status	STATUSI
Complejidad	Complexity	COMPLEXITYI

Statusi se define en ASYNC_DEF_STATE
Complexityi está definido en ASYNC_DEF_ROLE

Relacionado con la tarea ASYNC_TASK:

Ventana de parámetros:

- Dato: Tarea (estático)
- Estado (Status) Combobox ASYNC_TASK.statusi
    - ASYNC_DEF_STATUS_ERROR 4 “En error”
    - ASYNC_DEF_ STATUS_INACTIVE -1 “Inactiva”
    - TODOS “Todos”
    - ASYNC_DEF_ STATUS_WAITING 0 “A la espera de ser tratada”
    - ASYNC_DEF_ STATUS_SPLIT 1 “Tarea dividida en subtareas”
    - ASYNC_DEF_ STATUS_WORKING 2 “En curso”
    - ASYNC_DEF_ STATUS_FINISHED 3 “Terminada con éxito”
    - ASYNC_DEF_ STATUS_ABORTED 5 “Anulada”
- Por defecto ASYNC_DEF_ STATUS_ERROR
- Usuario (User) 31 caracteres Por defecto vacío
- Fecha de creación Date Por defecto DTSYSL
- Hora de salida Date Por defecto 0
- Hora de fin Date Por defecto 0

Si se introduce usuario, se filtra en función de TASK_LOGINS
La fecha es obligatoria por lo que filtramos gracias a DTCREAL = Fecha
Si la hora de salida introducida es 0, no se filtra ese campo
Si la hora de fin introducida es 0, no se filtra el campo HRENDL

Se mostrará un campo como el que se muestra a continuación:

Columna	Column	Campo
Id de la tarea	Task Id	ID_TASKL
Id tarea maestra	Master task Id	ASSOCIATED_TASKL
Estado	Status	STATUSI

Usuario	User	TASK_LOGINS
Id servicio web	Id web service	WEB_SERVICE_IDS
Tipo	Type	TASK_TYPEI
Fecha de creación	Date created	DTCREAL
Hora de creación	Hour created	HRCREAL
Fecha de fin	Date finished	DTENDL
Hora de fin	Hour finished	HRENDL
Duración (segundos)	Duration (sec)	DURATIONL
Objeto servidor	Server objet	TASK_MESS_TAGI
Tipo de consulta	Request type	TASK_TYPE_REQUESTI
Notificado	Notified	NOTIFIEDB
Numero de subtareas	Tot. Subtask	SUBTASK_NUMBERI
Nombre del host	Host names	SERVER_NAMES
Puerto	Port	SERVER_PORTL

Statusi está definido en ASYNC_DEF_STATUSI
Task_typei se define en ASYNC_TASK_TYPE
TASK_MESS_TAGI se define en object_mess_tagi.h
TASK_TYPEREQUESTI está definido en sysenv.h

Subtarea ASYNC_SUBTASK:

Ventana de parámetros:

- Dato: Subtarea (estático)
- Estado (status): Combobox ASYNK_SUBTASK.statusi
    - ASYNC_DEF_STATUS_ERROR 4 “En error”
    - ASYNC_DEF_ STATUS_INACTIVE -1 “Inactiva”
    - TODOS “Todos”
    - ASYNC_DEF_ STATUS_WAITING 0 “A la espera de ser tratada”
    - ASYNC_DEF_ STATUS_SPLIT 1 “Tarea dividida en subtareas”
    - ASYNC_DEF_ STATUS_WORKING 2 “En curso”
    - ASYNC_DEF_ STATUS_FINISHED 3 “Terminada con éxito”
    - ASYNC_DEF_ STATUS_ABORTED 5 “Anulada”
- Por defecto ASYNC_DEF_ STATUS_ERROR
- Por defecto ASYNC_DEF_ STATUS_ERROR
- Usuario (User) 31 caracteres Por defecto vacío
- Fecha de creación Date Por defecto DTSYSL
- Hora de salida Date Por defecto 0
- Hora de fin Date Por defecto 0

Si se introduce usuario se filtra gracias a TASK_LOGINS, por lo que se hace una unión entre las tablas ASYNC_TASK y ASYNC_SUBTASK
La fecha es obligatoria por lo que filtramos gracias a DTCREAL = Fecha de la tabla ASYNC_SUBTASK
Si la hora de salida introducida es 0, no se filtra ese campo de la tabla ASYNC_SUBTASK
Si la hora de fin introducida es 0, no se filtra el campo HRENDL de la tabla ASYNC_SUBTASK

Se mostrará una tabla con las columnas siguientes:

Columna	Column	Campo
Id de la tarea	Task Id	ID_TASKL
Número de la subtarea	Nb. Subtask	SUBTASK_NUMBERI
Número de subtareas	Tot. Subtask	SUBTASK_TOTALI
Estado	Status	STATUSI
Usuario	User	ASYNC_TASK.TASK_LOGINS
Nombre del proceso	Process name	CLIENT_NAMES
Id servicio web	Id web service	WEB_SERVICE_IDS
Tipo de la subtarea	Subtask type	SUBTASK_TYPEI
Fecha de creación	Date created	DTCREAL
Hora de creación	Hour created	HRCREAL
Fecha de inicio	Date start	DTSTARTL
Hora de inicio	Hour start	HRSTARTL
Fecha de fin	Date finished	DTENDL
Hora de fin	Hour finished	HRENDL
Duración (segundos)	Duration (sec)	DURATIONL
Complejidad	Complexity	COMPLEXITYI
Nombre del host	Host names	SERVER_NAMES
Puerto	Port	SERVER_PORTL

Statusi está definido en ASYNC_DEF_STATUSI
Complexityi está definido en ASYNC_DEF_ROLE
Subtask_typei se define en ASYNC_SUBTASK_TYPE

4.2 Realización del proyecto

Una vez analizado este documento con las especificaciones requeridas por el jefe de proyecto, comienza la reflexión sobre las diferentes etapas necesarias para resolver el proyecto al igual que el orden en el que se harán.

4.2.1 Función Display

Se decidió que era más interesante empezar por la función Display que al ser más complicada y completa permite entender desde el principio el funcionamiento y la comunicación cliente – servidor al igual que la manera con la que se integran las diferentes funcionalidades entre sí.

Lo primero que hay que desarrollar en una función de este tipo es la parte servidor. Esta parte se encarga de realizar la consulta a la base de datos. La primera tarea es por lo tanto la comprensión de las diferentes tablas de la base de datos necesarias al igual que la búsqueda de la estructura de las consultas SQL necesarias para conseguir la información requerida en cada caso.

Posteriormente se procede a escribir los programas necesarios para la parte servidor en lenguaje C, para ello se sigue un planteamiento bottom up, es decir empezando por abajo,

en este caso la base de datos, para llegar finalmente a la comunicación con el cliente compuesto en primer lugar por los APIJAVAS.

La primera etapa que se llevó a cabo fue la comprensión y el análisis de los programas existentes, buscando partes de código recurrentes que se deben incluir en todos los programas al igual que ejemplos que permiten entender cómo se realiza la comunicación con el cliente.

Una vez analizado, empieza la programación. En el caso de la función Display no hace falta crear nuevos objetos, simplemente completar programas ya existentes añadiéndoles la posibilidad de comunicarse con la parte cliente. Los programas que hay que modificar son async_task, async_subtask y asinc_registration, para cada caso el programa .c al igual que él archivo de cabecera .h respectivo.

Cada programa sirve para realizar las consultas a su tabla respectiva y en ambos casos las modificaciones y actualizaciones son muy parecidas. En los archivos .h lo que se añade es una estructura que va a contener las variables de cada tabla, de forma que se podrá transferir la tabla entera del lado servidor al lado cliente y a la inversa se podrán potencialmente realizar consultas desde el lado cliente en función de cualquiera de las variables que forman la tabla. Es importante que todos los elementos que vayan a usarse a ambos lados tengan el mismo nombre si no la comunicación no será posible. En el archivo .h también se añade la definición de algunas de las nuevas funciones que se escriben en el archivo .c respectivo. A continuación se muestra la estructura que se añade para el ejemplo del archivo de cabecera de las tareas, async_task.h.

```
+typedef struct __async_taskrequest__ /* Std Object query structure definition*/
+{
+    short               typerequesti    ;
+    short               tagi            ;
+
+    long                id_taskl;
+    long                associated_taskl;
+    short               statusi;
+    short               subtask_numberi;
+    short               task_mess_tagi;
+    short               task_typerequesti;
+    CDACCESS_ACCTELS    task_logins;
+    short               notifiedb;
+    long                dtcreal;
+    long                hrcreal;
+    long                timestamp_creal;
+    long                dtendl;
+    long                hrendl;
+    long                timestamp_endl;
+    long                durationl;
+    WEB_SERVICE_IDS     web_service_ids;
+    short               reservedb;
+    short               task_typei;
+} ASYNC_TASKREQUEST;
+
+

+ASYNC_TASK_EXTDEF void async_task_server_process(BLOCKPTR, SOCKET);
+ASYNC_TASK_EXTDEF void async_task_from_objectrequest(void *, SOCKET );
```

En el archivo .c primero se añade la definición de un tipo de estructura formada por varias macros. Una macro es un identificador que sustituye un trozo de código y que va a realizar una serie de instrucciones, en este caso permite codificar la información para enviarla y descodifica la información cuando se recibe. En dicha estructura se definen todas las variables de la tabla al igual que en el archivo .h, en este caso se hace de forma distinta y hay que especificar ciertos puntos además del tipo de objeto que es, en particular si existe realmente o no en la tabla, aquí nosotros al estar realizando la parte request ninguna variable existe realmente en ninguna tabla por lo que se le añade "extra" aunque posteriormente nos basemos en su tabla correspondiente donde sí que existe dicha variable.

```
1.  M_INIT_DESC2(ASYNC_TASKREQUEST, ASYNC_TASKREQUEST_ID, "PATIO_PATIO", "ASYNC_TASKREQ
    UEST", MESS_TAG_ASYNC_TASKREQUEST)
2. +
3. +    M_DATA_EXTR("TYPEREQUESTI",     SQL_SMALLINT, SHORT,            typerequesti)
4. +    M_DATA_EXTR("TAGI",             SQL_SMALLINT, SHORT,            tagi)
5. +
6. +    M_DATA_EXTR("ID_TASKL",         SQL_INTEGER,  LONG,             id_taskl)
7. +    M_DATA_EXTR("ASSOCIATED_TASKL", SQL_INTEGER,  LONG,             associated_task
    l)
8. +    M_DATA_EXTR("STATUSI",          SQL_SMALLINT, SHORT,            statusi)
9. +    M_DATA_EXTR("SUBTASK_NUMBERI",  SQL_SMALLINT, SHORT,            subtask_numberi
    )
10. +   M_DATA_EXTR("TASK_MESS_TAGI",   SQL_SMALLINT, SHORT,            task_mess_tagi)

11. +   M_DATA_EXTR("TASK_TYPEREQUSTI", SQL_SMALLINT, SHORT,            task_typereques
    ti)
12. +   M_DATA_EXTR("TASK_LOGINS",      SQL_CHAR,     CDACCESS_ACCTELS, task_logins)
13. +   M_DATA_EXTR("NOTIFIEDB",        SQL_SMALLINT, SHORT,            notifiedb)
14. +
15. +   M_DATA_EXTR("DTCREAL",          SQL_INTEGER,  LONG,             dtcreal)
16. +   M_DATA_EXTR("HRCREAL",          SQL_INTEGER,  LONG,             hrcreal)
17. +   M_DATA_EXTR("TIMESTAMP_CREAL",  SQL_INTEGER,  LONG,             timestamp_creal
    )
18. +   M_DATA_EXTR("DTENDL",           SQL_INTEGER,  LONG,             dtendl)
19. +   M_DATA_EXTR("HRENDL",           SQL_INTEGER,  LONG,             hrendl)
20. +   M_DATA_EXTR("TIMESTAMP_ENDL",   SQL_INTEGER,  LONG,             timestamp_endl)

21. +   M_DATA_EXTR("DURATIONL",        SQL_INTEGER,  LONG,             durationl)
22. +   M_DATA_EXTR("WEB_SERVICE_IDS",  SQL_CHAR,     WEB_SERVICE_IDS,  web_service_ids
    )
23. +   M_DATA_EXTR("RESERVEDB",        SQL_SMALLINT, SHORT,            reservedb)
24. +   M_DATA_EXTR("TASK_TYPEI",       SQL_SMALLINT, SHORT,            task_typei)
25. +
26. +
27. +   M_OBJECTREQUEST3_FP(async_task_from_objectrequest);
28. +
29. +   M_END_DESC()
30. +   M_END_DESC_CLI()
31. +   M_TABLE_NOT_IN_DATABASE()
32. +}
```

Esta es la estructura añadida para poder codificar y descodificar la información a la hora de realizar consultas de búsqueda de tareas asíncronas.

En el cuerpo de los tres programas se escriben las 3 funciones siguientes, que realizan lo mismo para cada tabla. La primera es, ASYNC_NOMBRE_DE_LA_TABLA_FROM_OBJECTREQUEST donde, en función del tipo de

consulta, en nuestro caso solo se entra si la consulta es de tipo lectura, le asocia el mess_tag de dicha tabla al igual que su ID y llama a la segunda función.

```
1.  /*
2.  +** ASYNC_TASK_FROM_OBJECTREQUEST
3.  +*/
4.  +
5.  +void async_task_from_objectrequest(void *datap, SOCKET socket)
6.  +{
7.  +    BLOCK block ={0};
8.  +    OBJECTREQUEST *objectrequestp = (OBJECTREQUEST *) datap;
9.  +
10. +    if (objectrequestp -> typerequesti == TYPEREQUEST_READ)
11. +    {
12. +        block.blockheader.blocktagi = MESS_TAG_ASYNC_TASKREQUEST;
13. +        block.datap = mem_get(database_desc[ASYNC_TASKREQUEST_ID]->datasize);
14. +        objectrequestp->id_tablei = ASYNC_TASKREQUEST_ID;
15. +    }
16. +
17. +    objectrequest_fill_request(objectrequestp, block.datap);
18. +    async_task_server_process(&block, socket);
19. +    mem_free(block.datap);
20. +}
```

La siguiente función, ASYNC_X_SERVER_PROCESS, analiza el mess_tag, si es el correcto, y en función del tipo de consulta llama a la función que realiza la acción que queremos. En este caso solo se quiere leer la tabla en la base de datos con lo cual solo se ha escrito una función que permite realizar dicha acción, ASYNC_X_SERVER_READ.

```
1.  *
2.  +** ASYNC_TASK_SERVER_PROCESS
3.  +*/
4.  +
5.  +void async_task_server_process(BLOCKPTR blockp, SOCKET socket)
6.  +{
7.  +    ASYNC_TASKREQUEST *async_taskrequestp =NULL;
8.  +    SHORT typerequesti=0;
9.  +
10. +    if (blockp->blockheader.blocktagi == MESS_TAG_ASYNC_TASKREQUEST)
11. +    {
12. +        async_taskrequestp = (ASYNC_TASKREQUEST *) blockp->datap;
13. +        typerequesti = async_taskrequestp->typerequesti;
14. +    }
15. +
16. +    switch (typerequesti)
17. +    {
18. +        case TYPEREQUEST_READ :
19. +            async_task_server_read(async_taskrequestp, socket);
20. +        break;
21. +        default :
22. +             seraudit_var("ERROR: %s typerequesti not handled",__func__);
23. +        break;
24. +    }
25. +}
```

Es en dicha función donde en función de los parámetros que ha recibido se crea la consulta SQL, aunque los parámetros varíen en función de la tabla el principio es el mismo, si se ha recibido un parámetro se incluirá en la consulta SQL, para ello es una estructura simple por ejemplo en el caso de una tarea si se ha recibido el estado y la fecha la consulta creada será:

SELECT *
FROM ASYNC_TASK
WHERE DTCREAL = “fecha” AND STATUSI = “estado”

El resultado es una cuadro con las columnas de la tabla y el número de líneas igual al de tareas que concuerden con la búsqueda. Para enviarlo al cliente, se crea un puntero en el que se almacena dicho cuadro.

```
1.  /*
2.  +** ASYNC_TASK_SERVER_READ
3.  +*/
4.  +
5.  +static void async_task_server_read(ASYNC_TASKREQUEST *async_taskrequestp, SOCKET so
    cket)
6.  +{
7.  +    char wheres[1000];
8.  +    char *ptr = NULL;
9.  +    NODEPTR invent =NULL;
10. +    long rowcountl = 0L;
11. +
12. +    seraudit_var("INFO: %s BEGIN ", __func__);
13. +
14. +    memset(wheres, '\0', sizeof(wheres));
15. +
16. +    ptr = wheres;
17. +
18. +    ptr += sprintf(ptr, "WHERE DTCREAL =%d", async_taskrequestp->dtcreal);
19. +
20. +    if (async_taskrequestp ->statusi < 99)
21. +    {
22. +        ptr += sprintf(ptr, " AND STATUSI =%d", async_taskrequestp ->statusi);
23. +    }
24. +
25. +    if (!st_isblank(async_taskrequestp ->task_logins))
26. +    {
27. +        ptr += sprintf(ptr, " AND TASK_LOGINS='%s'", async_taskrequestp -
    >task_logins);
28. +    }
29. +
30. +    if (async_taskrequestp ->hrendl != 0)
31. +    {
32. +        ptr += sprintf(ptr, " AND HRENDL <=%d", async_taskrequestp ->hrendl);
33. +    }
34. +
35. +    if (async_taskrequestp ->hrcreal != 0)
36. +    {
37. +        ptr += sprintf(ptr, " AND HRCREAL >=%d", async_taskrequestp ->hrcreal);
38. +    }
39. +
40. +    invent = node_list_init();
41. +
42. +    DBINVENT(ASYNC_TASK_ID, wheres, invent, &rowcountl);
43. +    seraudit_var("INFO: %s found %ld async tasks ", __func__, rowcountl);
```

```
44. +
45. +    object_server_invent_send_bundle(ASYNC_TASK_ID, NULL, invent, socket);
46. +
47. +    node_list_dispose(invent);
48. +    seraudit_var("INFO: %s END ", __func__);
49. +}
```

Tal y como se observa en el trozo de código anterior, para poder realizar la lectura en la base de datos se utiliza la macro DBINVENT. Se trata de una macro ya existente en el servidor de la empresa que se utiliza en el caso de consultas simples y básicas, sin embargo el código que hay detrás es bastante complicado de entender razón por la cual no se han incluido los detalles en este proyecto.

Esta es la dinámica general en la que se basa el servidor para esta función, sin embargo hay ciertas diferencias en función de si se trata de una subtarea o de otro de los datos estáticos. Dichas diferencias se encuentran en la función server_read, particularmente a la hora de construir la consulta SQL. La diferencia es que en el caso de una subtarea se quiere tener la posibilidad de poder filtrar en función del id de la tarea principal a partir de la cual se ha generado la sub tarea. Por definición este parámetro no está en la tabla de subtareas sino en la de tareas, por lo que es necesario realizar una unión (join) entre las tablas de datos. Se trata de un procedimiento muy habitual y básico en las consultas SQL por lo que no resulta especialmente complicado construir este tipo de consultas. Además este caso, al igual que el anterior en el que no es necesario realizar unión ya han sido realizados anteriormente y ya existe su macro correspondiente.

Una vez realizada esta parte y después de la comprobación por parte de un compañero del equipo de desarrollo se procedió a programar la primera capa de la parte cliente, los APIJAVA. Hasta ahora aunque el código no presentaba errores de compilación no había manera de comprobar que funcionaba según lo previsto.

Para las APIJAVA de la función Display se necesita la creación de 6 clases, esto se debe a que al contrario que en C, no se pueden juntar dos clases diferentes en el mismo archivo por mucho que sea lógico que vayan juntas. En este caso es especialmente complicado porque cada una de las clases va asociada con su propio mess_tagi e id de tabla. Además hasta el momento no había ningún tránsito de información sobre los servidores, tareas o subtareas entre el servidor y los servicios web por lo que es necesario crear toda la estructura en la parte cliente.

Se crea una clase para definir cada tabla, y su clase asociada, añadiéndole el sufijo REQUEST, que permite realizar la consulta.

En el primer tipo de clase se definen los parámetros de la tabla, hay que tener especial cuidado en utilizar los mismos nombres que del lado servidor, esto es válido para todo lo que concierne las APIJAVA, de lo contrario el programa no es capaz de unir las variables a ambos lados. Aunque no sea obligatorio es recomendable debido a que facilita la comprensión, sobre todo cuando el número de parámetros empieza a ser importante, utilizar siempre el mismo nombre para referirse a un mismo parámetro en las capas superiores. A continuación viene el

constructor que permite inicializar el objeto y comprobar que contiene los valores válidos. Finalmente como al final de todos los programas en JAVA se hallan los get/set.

En las clases de tipo request encontramos el mismo código que define los parámetros que anteriormente al igual que el constructor. Sin embargo se añaden dos métodos que permiten construir la consulta que será enviada a través del socket. La primera función se encarga de construir la consulta con los parámetros que el usuario ha introducido y que le pasa la segunda función que se muestra más adelante. El código siguiente pertenece a la clase ASYNC_SUBTASKREQUEST, que gestiona las consultas de subtareas asíncronas.

```
1.  +    private static OBJECTREQUEST initAsync_SubTaskRequest(short statusi, String tas
    k_logins, long dtcreal, long hrcreal, long hrendl, long id_taskl)
2.  +    {
3.  +       OBJECTREQUEST objectrequest = new OBJECTREQUEST(new ASYNC_SUBTASKREQUEST());

4.  +       objectrequest.setTypeRequesti(SYSENV.TYPEREQUEST_READ);
5.  +       objectrequest.setParam("statusi", statusi);
6.  +       objectrequest.setParam("task_logins", task_logins);
7.  +       objectrequest.setParam("dtcreal", dtcreal);
8.  +       objectrequest.setParam("hrcreal", hrcreal);
9.  +       objectrequest.setParam("hrendl", hrendl);
10. +       objectrequest.setParam("id_taskl", id_taskl);
11. +
12. +       return objectrequest;
13. +    }
```

La segunda función obtiene los parámetros introducidos por el usuario y se los pasa a la función anterior. Posteriormente crea la socket para comunicarse con el servidor y gestiona la recopilación del resultado recibido. Asimismo se muestra el ejemplo en el caso de las subtareas.

```
1.  +    public static List ReadFromServer(PatioSocket patiosocket, short statusi, Strin
    g task_logins, long dtcreal, long hrcreal, long hrendl, long id_taskl) throws PatioE
    xception
2.  +    {
3.  +       OBJECTREQUEST objectrequest = initAsync_SubTaskRequest(statusi, task_logins,
     dtcreal, hrcreal, hrendl, id_taskl);
4.  +
5.  +       ActioList list = objectrequest.getActioList(patiosocket);
6.  +
7.  +       List result = new ArrayList();
8.  +       for (int i = 0; i < list.getList().size(); i++)
9.  +        {
10. +            Object elt = list.getList().get(i);
11. +            ASYNC_SUBTASK newAsync_SubTask = (ASYNC_SUBTASK) elt;
12. +            result.add(newAsync_SubTask);
13. +        }
14. +
15. +              return result;
16. +    }
```

Llegados a este punto se tiene por primera vez la posibilidad de probar el correcto funcionamiento del código escrito hasta ahora. Para ello se dispone de una clase de test en la

que hay una pequeña interfaz muy básica que se conecta al servidor de forma local y permite realizar las pruebas. Para eso, en la empresa, han creado un switch case en el que se llama a la clase que se quiere testear, se pide al usuario que rellene los parámetros que necesita y se construye la consulta del mismo modo que se haría realmente, lo que permite ver su funcionamiento.

A continuación se muestra el diálogo en el caso que se quiera probar el funcionamiento del código escrito para la búsqueda de tareas.

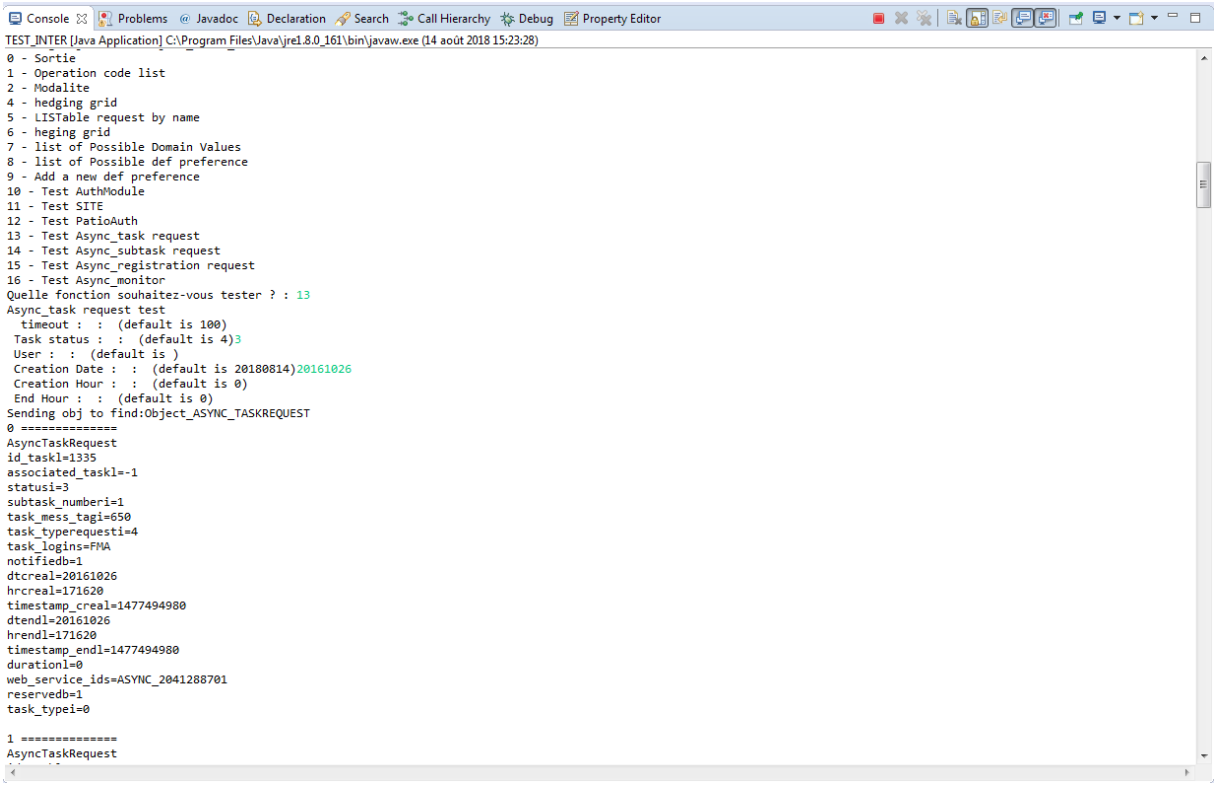


Figura 8: Diálogo de prueba en el caso de una consulta de búsqueda a la tabla de tareas

Para poder comprobar de forma más sencilla y visual, en la clase asociada a la tabla, es decir ASYNC_TASK se añadirá una pequeña función toString que permita ver directamente el resultado. En este caso se puede ver que cuando se realiza la búsqueda de una tarea terminada correctamente, creada el 26/10/2016 tenemos varios resultados. En la captura se muestra el primero y se puede ver que todas las informaciones relacionadas con dicha tarea aparecen correctamente.

```
1.   public String toString()
2.  +    {
3.  +       StringBuffer sb = new StringBuffer();
4.  +
5.  +       sb.append("AsyncRegistrationRequest\n");
6.  +       sb.append(ASYNC_REGISTRATION.SERVER_NAMES + "=" + server_names + "\n");
7.  +       sb.append(ASYNC_REGISTRATION.SERVER_PORTL + "=" + server_portl + "\n");
8.  +       sb.append(ASYNC_REGISTRATION.CLIENT_NAMES + "=" + client_names + "\n");
```

```
9.  +       sb.append(ASYNC_REGISTRATION.HEARTBEAT_DATEL + "=" + heartbeat_datel + "\n")
    ;
10. +       sb.append(ASYNC_REGISTRATION.HEARTBEAT_HOURL + "=" + heartbeat_hourl + "\n")
    ;
11. +       sb.append(ASYNC_REGISTRATION.HEARTBEAT_TIMESTAMPL + "=" + heartbeat_timestam
    pl + "\n");
12. +       sb.append(ASYNC_REGISTRATION.STATUSI + "=" + statusi +"\n");
13. +       sb.append(ASYNC_REGISTRATION.COMPLEXITYI + "=" + complexityi + "\n");
14. +       sb.append(ASYNC_REGISTRATION.USE_PREALLOCATIONB + "=" + use_preallocationb +
     "\n");
15. +       return sb.toString();
```

Esta es la función toString utilizada para comprobar visualmente el resultado de una consulta a la tabla de servidores.

Una vez que se ha comprobado el correcto funcionamiento para las tres tablas y con todas las combinaciones posibles de parámetros utilizados se procede a la programación de la capa superior.

A partir de aquí se procede utilizando una metodología Top – down, contrariamente al planteamiento utilizado anteriormente, esto supone que se parte del resultado final en nuestro caso el desarrollo de los componentes que gráficamente aparecen en la interfaz y con los que el usuario realiza las búsquedas y obtiene los resultados para llegar finalmente a la unión con las APIJAVAS. La elección de esta metodología fue una sugerencia de un compañero debido a que técnicamente resulta más sencillo a causa de ciertas particularidades que la empresa usa a la hora de transformar los datos con el objetivo de poder utilizarlos en las diferentes capas que conforman la estructura cliente.

Esta parte, aunque no es necesariamente más compleja de programar sí que requiere la realización de un número muy importante de clases, bien es cierto que no suelen ser muy largas. La mayoría de las clases están relacionadas entre sí, por lo que al crear una te sugiere la creación de la siguiente advirtiéndote de que falta, además es un procedimiento bastante mecánico, se deben crear más o menos las mismas clases para cualquier tipo de funcionalidad con algunas variaciones en función de donde serán implantadas y de sus particularidades.

En el caso de este proyecto en particular no había ninguna funcionalidad con las mismas características por lo que hubo que improvisar en ciertos momentos, no obstante gran parte de las clases se puede encontrar de forma parecida en los archivos de funcionalidades anteriores.

A continuación se explica brevemente la utilidad de cada uno de los programas que se realizaron.

Primero hay que definir los objetos que van a ser necesarios en las diferentes capas del cliente, se empieza creando un objeto AsyncMonitorDisplay en el cual se definen las variables que contiene, process para los servidores, task para las tareas y subtask para las subtareas al igual que un método que permita saber que variable se ha seleccionada tal y como se ve en este trozo de código.

```
1.   public AsyncMonitorDisplay(String title, boolean process, boolean task, boolean sub
     task)
2.  +    {
3.  +        super(title);
4.  +        this.process = process;
5.  +        this.task = task;
6.  +        this.subtask = subtask;
7.  +    }
8.  +
9.  +    @Override
10. +    public String getIdentificator()
11. +    {
12. +        return this.getTitle();
13. +    }
14. +
15. +    public boolean isTask()
16. +    {
17. +        return task;
18. +    }
19. +
20. +    public void setTask(boolean task)
21. +    {
22. +        this.task = task;
23. +    }
24. +
25. +    public boolean isSubtask()
26. +    {
27. +        return subtask;
28. +    }
29. +
30. +    public void setSubtask(boolean subtask)
31. +    {
32. +        this.subtask = subtask;
33. +    }
34. +
35. +    public boolean isProcess()
36. +    {
37. +        return process;
38. +    }
39. +
40. +    public void setProcess(boolean process)
41. +    {
42. +        this.process = process;
43. +    }
```

A continuación es necesario un objeto que define cada una de dichas variables, es decir AsyncMonitorTask, AsyncMonitorSubtask y AsyncMonitorProcess. En ellos se definen únicamente los campos que componen los objetos que no son más que los que aparecen en la tabla de la base de datos. Posteriormente puesto que se van a realizar consultas es necesaria, al igual que en las APIJAVA, una clase Request para cada una de las clases anteriores en donde se incluyen en este caso no todos los campos sino únicamente los que utilizaremos para realizar las consultas. Por ejemplo en el caso de una consulta en la tabla de servidores, los únicos campos con los que se puede filtrar son el estado y el último heartbeat con lo cual solo se definen estos campos en la clase AsyncMonitorProcessRequest. En ambos casos al final de la clase se incluyen los get/set de cada variable para poder definirlos correctamente.

Es necesaria a continuación la creación de un grupo de clases con las que gestionar la apertura, una vez que se pulsa sobre una de las categorías, servidores, tareas o subtareas, la creación de la ventana de parámetros y la de la ventana de resultados. Hay que crear este

grupo de clases para cada categoría siguiendo el mismo principio, simplemente incluyendo las particularidades que se quiere en cada uno.

La primera, OpenAsyncMonitorDisplay seguido del nombre de la categoría, gestiona la apertura de la pantalla, se trata de una clase muy corta en la que lo único que cambia entre una clase y otra es el nombre del editor y de la función.

La clase siguiente es de las más importantes, DisplayAsyncMonitor___Editor. Se trata de la clase con la que se crea la ventana de parámetros, se crea la consulta, se invoca el servicio que se encarga de transmitirla a las APIJAVA al igual que se rellenan ciertas funciones como si se quiere un subtítulo, si habrá ventana de resultados etc. A continuación se muestra el ejemplo de construcción de la ventana de parámetros de búsqueda de servidores que es la más sencilla.

```
1.  protected void buildParametersInputZoneContent(Composite parent, FormToolkit toolkit
    )
2.  +    {
3.  +        // Combobox
4.  +        parent.setLayout(new GridLayout());
5.  +        Composite ComboComposite = IhmUtilHolder.getIhmUtil().createComposite(paren
    t, SWT.NONE, "NoBackground");
6.  +        gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
7.  +        gridData.horizontalIndent = 40;
8.  +        ComboComposite.setLayoutData(gridData);
9.  +        ComboComposite.setLayout(new GridLayout(2, false));
10. +
11. +        Label status = IhmUtilHolder.getIhmUtil().createLabel(ComboComposite, SWT.N
    ONE,
12. +                I18nUtil.getI18nMessage("asyncmonitor.process.combo.title"), null,
    null);
13. +
14. +        gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
15. +        status.setLayoutData(gridData);
16. +
17. +        combo = (TPMEnumCombo<AsyncMonitorProcessStatus>) IhmUtilHolder.getIhmUtil(
    ).createTPMCombo(ComboComposite, SWT.NONE,
18. +                AsyncMonitorProcessStatus.values(), false, false, null, null);
19. +
20. +        gridData = new GridData(SWT.FILL, SWT.CENTER, true, false);
21. +        combo.setLayoutData(gridData);
22. +        combo.enableTooltips();
23. +        gridData = new GridData(SWT.FILL, SWT.CENTER, true, false);
24. +        gridData.widthHint = WIDTH;
25. +        gridData.horizontalIndent = INDENT;
26. +        combo.setLayoutData(gridData);
27. +        combo.setSelectedValue(AsyncMonitorProcessStatus.ZOMBIE);
28. +
29. +        // Last heartbeat
30. +
31. +        parent.setLayout(new GridLayout());
32. +        Composite heartbeatComposite = IhmUtilHolder.getIhmUtil().createComposite(p
    arent, SWT.NONE, "NoBackground");
33. +        gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
34. +        gridData.horizontalIndent = 40;
35. +        heartbeatComposite.setLayoutData(gridData);
36. +        heartbeatComposite.setLayout(new GridLayout(2, false));
37. +
38. +        heartbeatLabel = IhmUtilHolder.getIhmUtil().createLabel(heartbeatComposite,
     SWT.NONE,
```

```
+                I18nUtil.getI18nMessage("display.asyncmonitor.heartbeat"), null, null);
+        gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
+        heartbeatLabel.setLayoutData(gridData);
+
+        heartbeatsearch = IhmUtilParameter.createTPMText(heartbeatComposite, IhmUtil.TEXT_ALPHA_NUMERIC, null, null, 50, null);
+        GridData gr = new GridData(SWT.FILL, SWT.CENTER, true, false);
+        gr.widthHint = WIDTH;
+        heartbeatsearch.setLayoutData(gr);
+
+    }
```

A continuación se muestra como se construye la consulta también en el caso de la búsqueda de servidores,

```
+    protected void performValidate()
+    {
+        query = new AsyncMonitorProcessRequest();
+
+        query.setStatus(combo.getSelectedValue());
+
+        if (!heartbeatsearch.getText().isEmpty())
+        {
+            long heartbeat_timestampl = Long.parseLong(heartbeatsearch.getText());

+            query.setLast_heartbeat(heartbeat_timestampl);
+        }
+    }
```

Y finalmente como se invoca el servicio que envía la consulta a las APIJAVA.

```
+    public void invokeServiceOnValidate(TPMUserInfo userInfo) throws TPMServiceException
+    {
+        asyncMonitorProcesses = ServiceFactory.getBlockOrderService().asyncMonitorProcess(userInfo, query);
+
+    }
```

Para esta aplicación, puesto que está relacionada con las órdenes en bloque se ha incluido en una clase ya existente llamada BlockOrderService. En dicha clase simplemente se incluye la definición de las funciones, una para cada categoría, y es en la clase BlockOrderServiceImpl donde se escribe el contenido de cada función. A continuación se muestra el ejemplo para los servidores.

```
public List<AsyncMonitorProcess> doInPatio(ActioRequestBroker arb) throws TPMServiceException
+            {
+                arbTemplate.setStart(System.currentTimeMillis());
+
+                OBJECTREQUEST objectrequest = new OBJECTREQUEST(new ASYNC_REGISTRATIONREQUEST());
```

```
+                objectrequest.setTypeRequesti(SYSENV.TYPEREQUEST_READ);
+                if (query != null)
+                {
+                    objectrequest.setParam("statusi", query.getStatus().getValue())
;
+                    if (query.getLast_heartbeat() != 0)
+                        objectrequest.setParam("heartbeat_timestampl", query.getLas
t_heartbeat());
+                }
+
+                arbTemplate.setBeforeCallingPatio(System.currentTimeMillis());
+                ActioList result;
+                try
+                {
+                    result = arb.getActioList(objectrequest, (int) arb.getTimeout()
);
+                }
+                catch (PatioException e)
+                {
+                    TPMServiceException exc;
+                    exc = CommonServiceImpl.defineServiceException(e);
+                    throw exc;
+                }
+                arbTemplate.setAfterCallingPatio(System.currentTimeMillis());
+
+                // Conversion result api object in service object
+                AsyncMonitorProcessAdapter adapter = new AsyncMonitorProcessAdapter
();
+                List<AsyncMonitorProcess> asyncMonitorProcesses = new ArrayList<Asy
ncMonitorProcess>();
+                Iterator it = result.getIterator();
+                while (it.hasNext())
+                    asyncMonitorProcesses.add(adapter.convertFromPatioObj((APIJAVAO
BJSTD) it.next(), userInfo));
+
+                arbTemplate.setStop(System.currentTimeMillis());
+                SharedUtil.trace(LOGGER, arbTemplate.getStart(), arbTemplate.getBef
oreCallingPatio(),
+                        arbTemplate.getAfterCallingPatio(), arbTemplate.getStop(),
"asyncMonitorProcess",
+                                "BlockOrderServiceImpl", arb.getServerName(), arb.getSe
rverPort(), arb.getCurrentTransactionId());
+
+                return asyncMonitorProcesses;
```

Esta función es la que va a rellenar el objeto de las APIJAVA con nuestra consulta y va a realizar la conversión entre el resultado que obtienen las APIJAVA y el objeto de las capas superiores, el servicio web.

La última función de este grupo se encarga de crear la ventana de resultados, DisplayAsyncMonitor___Result. Es una clase parecida a la clase Editor donde se construye la apariencia gráfica de la ventana de resultados y donde se rellenan ciertas funciones básicas relacionadas con títulos y demás. En este caso se muestra el ejemplo para la creación de dicha ventana para la clase de subtareas.

```
+    protected void buildResultContent(Composite parent, List<AsyncMonitorSubtask> d
ata, FormToolkit toolkit, SWTViewer swtViewer)
+    {
+        TableData tableData = new TableData(3216L, AsyncMonitorDisplaySubtaskDataSe
t.class.getName(), "tableAsyncMonitorSubtask", null);
```

```
4. +        Composite composite = IhmUtilHolder.getIhmUtil().createComposite(parent, SW
   T.NONE, null);
5. +        composite.setLayout(new GridLayout());
6. +        swtViewer.buildTableContent(composite, tableData);
7. +
8. +    }
```

El código 3216L es el código de la tabla que se obtiene al crear la clase de tipo rptdesign que permite exportar la tabla de resultados a diferentes formatos, en particular a pdf. La realización de estas clases es un poco particular puesto que se hace en XML aunque hay una vista especial en eclipse que permite hacerlo de manera más gráfica.

Además del código se especifica el nombre de la clase que hace de DataSet, en ella se escriben dos funciones que van a permitir la construcción de la tabla de resultados. La primera función se encarga de la creación de las columnas añadiéndoles el nombre de la columna y especificando de que tipo va a ser el dato que se le va a introducir mientras que la segunda se encarga de rellenar la celdas con el dato. A continuación se muestra el ejemplo de dichas funciones también en el caso de una subtarea,

```
1.  @Override
2.  +    protected List<ReportColumnHolder> getColumns(String tableName)
3.  +    {
4.  +        reportColumnHolders
5.  +                .add(new ReportColumnHolder("taskid", "async.monitor.subtask.taskid
    ", TPMColumnType.INTEGER, true, true));
6.  +        reportColumnHolders
7.  +                .add(new ReportColumnHolder("nbsubtask", "async.monitor.subtask.nbs
    ubtask", TPMColumnType.INTEGER, true, true));
8.  +        reportColumnHolders
9.  +                .add(new ReportColumnHolder("totsubtask", "async.monitor.subtask.to
    tsubtask", TPMColumnType.INTEGER, true, true));
10. +        reportColumnHolders.add(new ReportColumnHolder("status", "async.monitor.sub
    task.status", TPMColumnType.STRING, true, true));
11. +        reportColumnHolders.add(new ReportColumnHolder("user", "async.monitor.subta
    sk.user", TPMColumnType.STRING, true, true));
12. +        reportColumnHolders
13. +                .add(new ReportColumnHolder("processname", "async.monitor.subtask.p
    rocessname", TPMColumnType.STRING, true, true));
14. +        reportColumnHolders.add(
15. +                new ReportColumnHolder("idwebservice", "async.monitor.subtask.idweb
    service", TPMColumnType.STRING, true, true));
16. +        reportColumnHolders.add(
17. +                new ReportColumnHolder("subtasktypei", "async.monitor.subtask.subta
    sktypei", TPMColumnType.STRING, true, true));
18. +        reportColumnHolders
19. +                .add(new ReportColumnHolder("dtcreated", "async.monitor.subtask.dtc
    reated", TPMColumnType.DATE, true, true));
20. +        reportColumnHolders
21. +                .add(new ReportColumnHolder("hrcreated", "async.monitor.subtask.hrc
    reated", TPMColumnType.TIME, true, true));
22. +        reportColumnHolders
23. +                .add(new ReportColumnHolder("dtstarted", "async.monitor.subtask.dts
    tarted", TPMColumnType.DATE, true, true));
24. +        reportColumnHolders
25. +                .add(new ReportColumnHolder("hrstarted", "async.monitor.subtask.hrs
    tarted", TPMColumnType.TIME, true, true));
26. +        reportColumnHolders
27. +                .add(new ReportColumnHolder("dtfinished", "async.monitor.subtask.dt
    finished", TPMColumnType.DATE, true, true));
28. +        reportColumnHolders
```

```
29. +                .add(new ReportColumnHolder("hrfinished", "async.monitor.subtask.hr
    finished", TPMColumnType.TIME, true, true));
30. +        reportColumnHolders
31. +                .add(new ReportColumnHolder("duration", "async.monitor.subtask.dura
    tion", TPMColumnType.INTEGER, true, true));
32. +        reportColumnHolders
33. +                .add(new ReportColumnHolder("complexity", "async.monitor.subtask.co
    mplexity", TPMColumnType.INTEGER, true, true));
34. +        reportColumnHolders
35. +                .add(new ReportColumnHolder("hostnames", "async.monitor.subtask.hos
    tnames", TPMColumnType.STRING, true, true));
36. +        reportColumnHolders.add(new ReportColumnHolder("port", "async.monitor.subta
    sk.port", TPMColumnType.INTEGER, true, true));
37. +
38. +        return reportColumnHolders;
39. +    }
40. +
41.
42. +    protected void performFetch(IUpdatableDataSetRow row, AsyncMonitorSubtask item)
     throws ScriptException
43. +    {
44. +        setRowColumnValue(row, "taskid", item.getId_taskl(), item.getClass().getSim
    pleName());
45. +        setRowColumnValue(row, "nbsubtask", item.getId_numberi(), item.getClass().g
    etSimpleName());
46. +        setRowColumnValue(row, "totsubtask", item.getSubtask_totali(), item.getClas
    s().getSimpleName());
47. +        setRowColumnValue(row, "status", I18nUtil.getI18nLabel(item.getStatus()), i
    tem.getClass().getSimpleName());
48. +        setRowColumnValue(row, "user", item.getTask_logins(), item.getClass().getSi
    mpleName());
49. +        setRowColumnValue(row, "processname", item.getClient_names(), item.getClass
    ().getSimpleName());
50. +        setRowColumnValue(row, "idwebservice", item.getWeb_service_ids(), item.getC
    lass().getSimpleName());
51. +        setRowColumnValue(row, "subtasktypei", I18nUtil.getI18nLabel(item.getSubtas
    k_typei()), item.getClass().getSimpleName());
52. +        setRowColumnValue(row, "dtcreated", item.getDtcreal(), DateType.DATE, item.
    getClass().getSimpleName());
53. +        setRowColumnValue(row, "hrcreated", item.getDtcreal(), DateType.TIME_HHMMSS
    , item.getClass().getSimpleName());
54. +        setRowColumnValue(row, "dtstarted", item.getDtstartl(), DateType.DATE, item
    .getClass().getSimpleName());
55. +        setRowColumnValue(row, "hrstarted", item.getDtstartl(), DateType.TIME_HHMMS
    S, item.getClass().getSimpleName());
56. +        setRowColumnValue(row, "dtfinished", item.getDtendl(), DateType.DATE, item.
    getClass().getSimpleName());
57. +        setRowColumnValue(row, "hrfinished", item.getDtendl(), DateType.TIME_HHMMSS
    , item.getClass().getSimpleName());
58. +        setRowColumnValue(row, "duration", item.getDurationl(), item.getClass().get
    SimpleName());
59. +        setRowColumnValue(row, "complexity", item.getComplexityi(), item.getClass()
    .getSimpleName());
60. +        setRowColumnValue(row, "hostnames", item.getServer_names(), item.getClass()
    .getSimpleName());
61. +        setRowColumnValue(row, "port", item.getServer_portl(), item.getClass().getS
    impleName());
62. +
63. +    }
```

Como se ha dicho anteriormente, es necesaria una clase que haga la unión entre las variables en las APIJAVAS y entre las capas superiores, dichas clases hacen de adaptadores entre unas variables y otras y simplemente igualan las dos variables. A continuación se

muestra AyncMonitorTask que es el ejemplo de este adaptador para las tareas que lo que hace es poner el valor de la variable en las APIJAVA dentro de la variable de las capas superiores.

```
1.  public AsyncMonitorTask convertFromPatioObj(APIJAVAOBJSTD patioObj, TPMUserInfo user
    Info)
2.  +    {
3.  +        ASYNC_TASK async_TASK = (ASYNC_TASK) patioObj;
4.  +        AsyncMonitorTask task = new AsyncMonitorTask();
5.  +
6.  +        task.setId_taskl(async_TASK.getId_taskl());
7.  +        task.setAssociated_taskl(async_TASK.getAssociated_taskl());
8.  +        task.setStatus(AsyncMonitorTaskStatus.getEnum(async_TASK.getStatusi()));
9.  +        task.setSubtask_numberi(async_TASK.getSubtask_numberi());
10. +        task.setTask_mess_tagi(async_TASK.getTask_mess_tagi());
11. +        task.setTask_typerequesti(async_TASK.getTask_typerequesti());
12. +        task.setTask_logins(async_TASK.getTask_logins());
13. +        task.setNotifiedb(async_TASK.getNotifiedb() == 1);
14. +        task.setDtcreal(
15. +                new TPMDate(async_TASK.getDtcreal(), async_TASK.getHrcreal(), DateT
    ype.DATE_TIME, CONFIG.getServerTimeZone()));
16. +        task.setDtendl(new TPMDate(async_TASK.getDtendl(), async_TASK.getHrendl(),
    DateType.DATE_TIME, CONFIG.getServerTimeZone()));
17. +        task.setDurationl(async_TASK.getDurationl());
18. +        task.setWeb_service_ids(async_TASK.getWeb_service_ids());
19. +        task.setReservedb(async_TASK.getReservedb() == 1);
20. +        task.setTask_typei(AsyncMonitorTaskType.getEnum(async_TASK.getTask_typei())
    );
21. +
22. +        return task;
23. +    }
```

Una vez que estas clases han sido creadas hay que juntarlas mediante extensiones para que sean visibles en la plataforma. Para ello hay un plug-in donde se definen dichas extensiones, también se puede hacer escribiendo directamente el plug-in en XML lo que en algunos casos resulta más sencillo. En la figura siguiente se ve la pantalla principal de dicho plug-in.

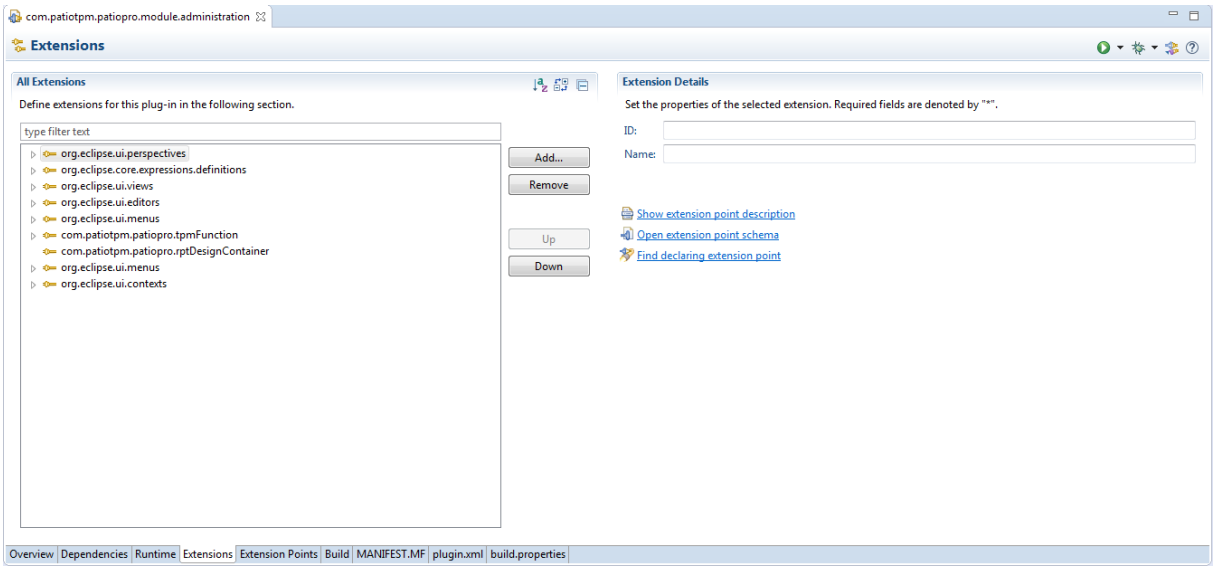


Figura 9: Pantalla principal del plug-in de extensiones

Se añaden extensiones en las categorías definición, editor, función y contexto.

En la categoría definición simplemente se definen ciertos elementos que van a ser necesarios para el correcto funcionamiento de las funcionalidades, a continuación se muestran dichos elementos.

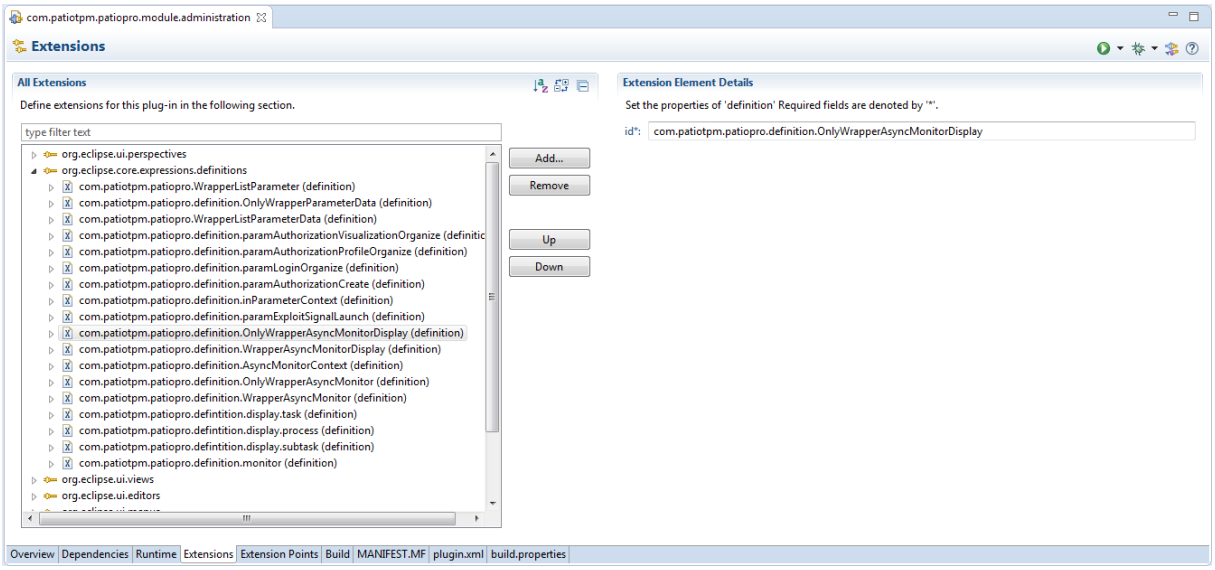


Figura 10: Categoría de definición del plug-in de extensiones

Más adelante se añaden las extensiones de los editores donde simplemente se asocia a cada extensión la clase en la que viene el editor que es donde por ejemplo se crea la ventana de parámetros al igual que un código de identificación.

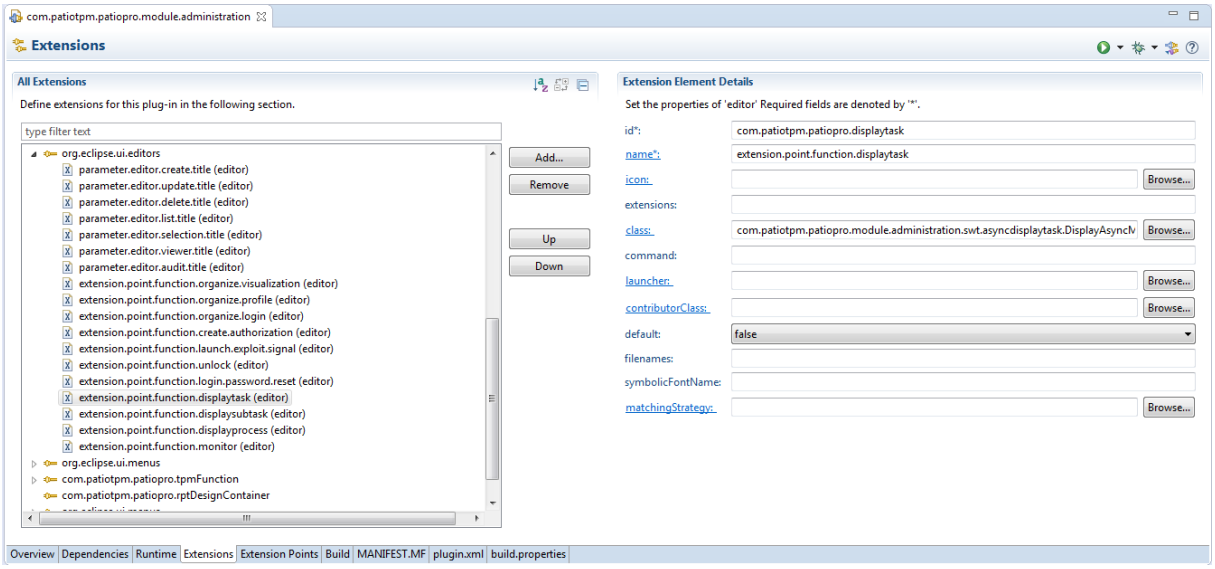


Figura 11: Categoría de editores del plug-in de extensiones

Una de las extensiones más significativas es la de funciones en ella se realizan varias acciones. Primero, igual que en el caso anterior, se asocia a la extensión un código de identificación, un nombre, la clase en la que se encuentra el editor y por último la imagen que queremos que se muestre en la plataforma al lado del nombre de la función.

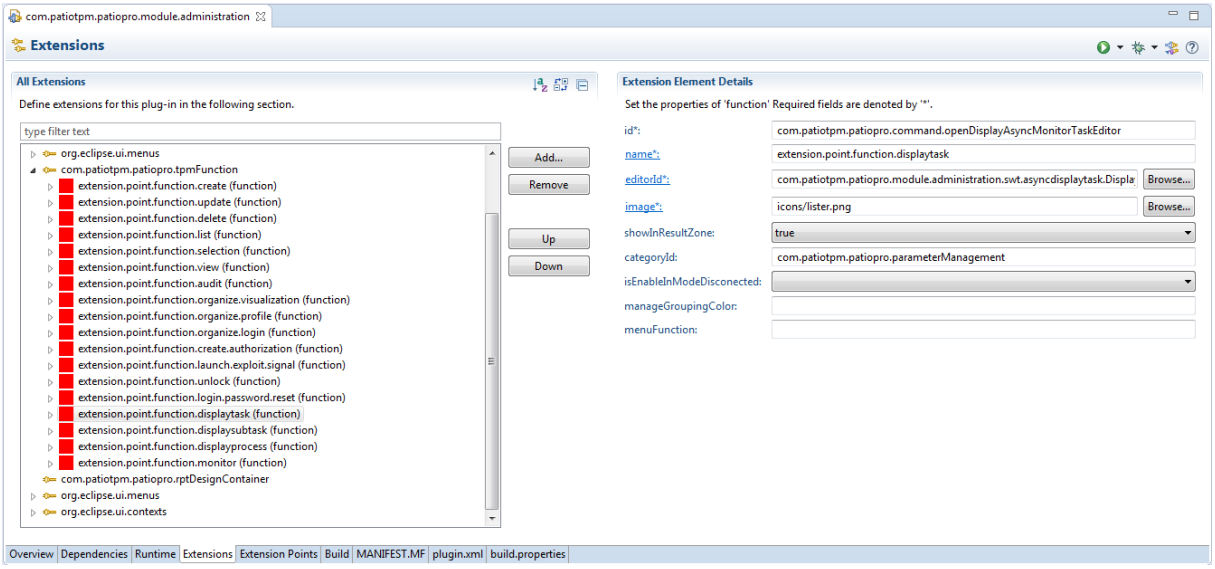


Figura 12: Categoría de funciones del plug-in de extensiones

Sin embargo además de esto hay que definir cuál será la clase encargada de abrir la funcionalidad y sobre todo dos cuestiones muy importantes, en que contextos se quiere que la funcionalidad sea visible y esté disponible.

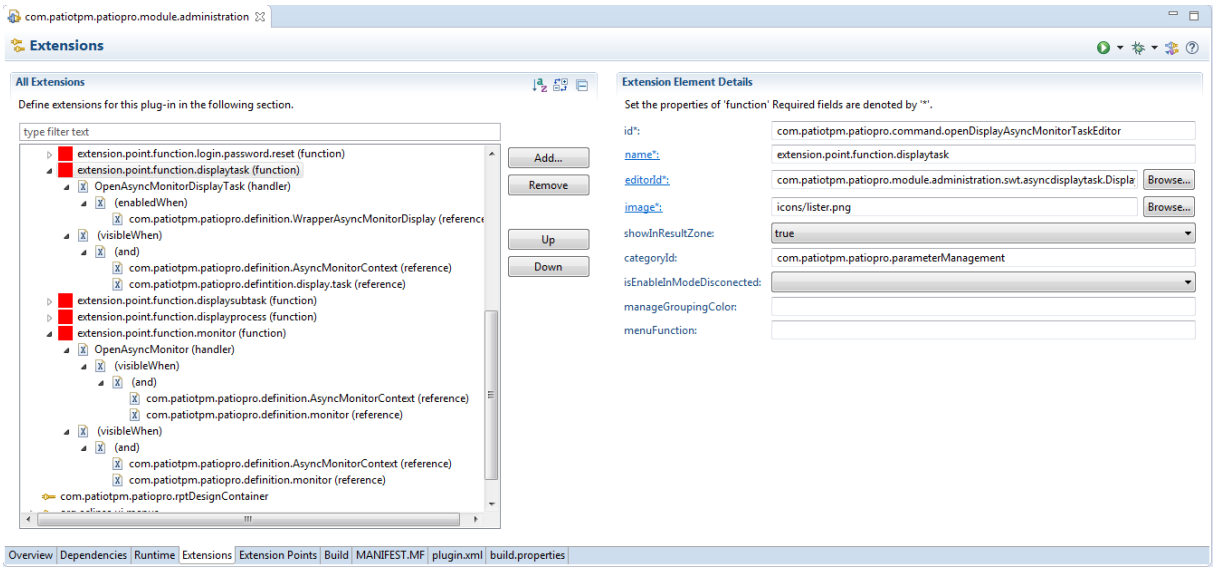


Figura 13: Categoría de funciones extendida del plug-in de extensiones

Esta parte es más sencilla de realizar escribiendo directamente en XML, a continuación se muestra el código para definir la extensión en el caso de una tarea.

```
      <function
            categoryId="com.patiotpm.patiopro.parameterManagement"

editorId="com.patiotpm.patiopro.module.administration.swt.asyncdisplaytask.Display
AsyncMonitorTaskEditor"
            id="com.patiotpm.patiopro.command.openDisplayAsyncMonitorTaskEditor"
            image="icons/lister.png"
            name="extension.point.function.displaytask"
            showInResultZone="true">
         <handler

class="com.patiotpm.patiopro.module.administration.swt.asyncdisplaytask.OpenAsyncM
onitorDisplayTask">
            <enabledWhen>
               <reference

definitionId="com.patiotpm.patiopro.definition.WrapperAsyncMonitorDisplay">
               </reference>
            </enabledWhen>
         </handler>
         <visibleWhen>
               <and>
                 <reference

definitionId="com.patiotpm.patiopro.definition.AsyncMonitorContext">
                 </reference>
                 <reference

definitionId="com.patiotpm.patiopro.defintition.display.task">
                 </reference>
               </and>
         </visibleWhen>
      </function>
```

Se quiere por lo tanto que la función sea visible bajo dos condiciones, primero cuando nos encontremos en la categoría Async Monitor y segundo cuando la categoría seleccionada sea una tarea. La función debe estar disponible cuando una de las categorías este seleccionada

Finalmente se define el contexto que es simplemente especificar que estamos en la categoría Async Monitor.

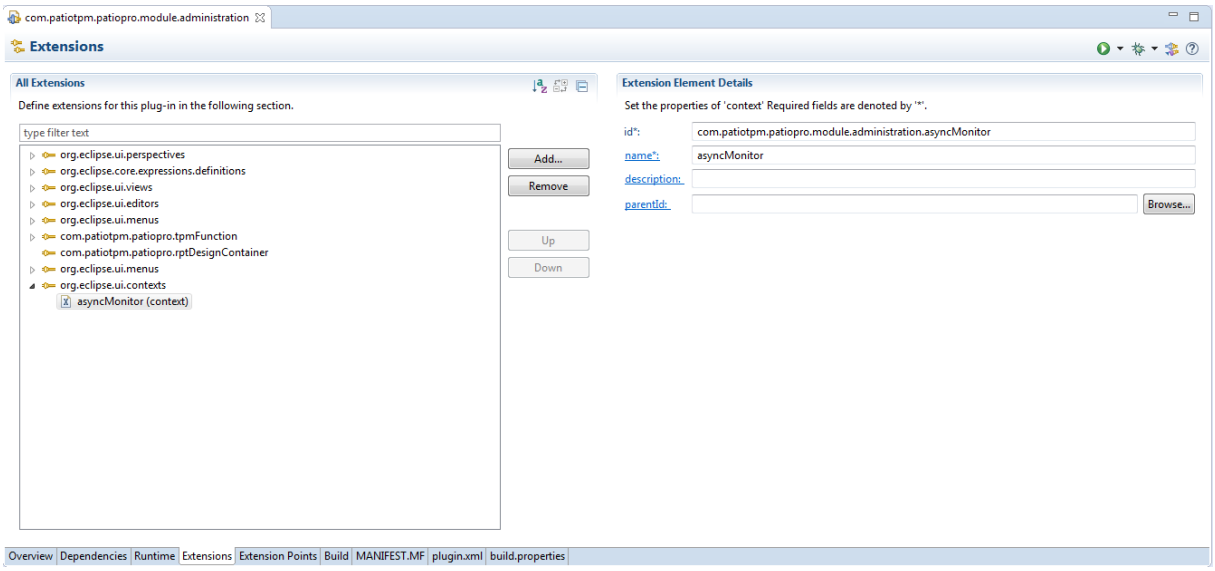


Figura 14: Categoría de contexto del plug-in de extensiones

Finalmente además de estas clases se han creado algunas más que estarán disponibles en Anexo donde se definen características, como por ejemplo los nombres de los estados. Asimismo hay clases necesarias para el funcionamiento de cualquier funcionalidad pero que no tienen especial interés para el desarrollo del proyecto, son clases cortas, generales para las tres categorías y cuyo único objetivo es el correcto funcionamiento de la funcionalidad.

Una vez que todas estas clases y archivos tanto en C como en Java han sido realizadas la función Display está terminada y operativa.

4.2.2 Función Monitor

Una vez la función Display completada se pasó a la realización del código correspondiente a la función Monitor. Dicha función realiza una consulta a la base de datos y requiere de un nuevo objeto en el servidor puesto que nunca se ha realizado nada parecido en la empresa. Este objeto, llamado ASYNC_MONITOR, busca los estados en las 3 tablas, de servidores, de tareas y de subtareas, los cuenta y los muestra en una tabla. La tabla no se actualiza sola sino que es necesario volver a pedir la información, lo que reenviará una consulta.

Siendo una función que se basa en las mismas tablas y con un funcionamiento muy parecido al de la función Display a pesar de ciertas particularidades, su estructura será también muy parecida a la de la función anterior.

Se procede como en el caso anterior empezando por la programación del servidor. En este caso toda la parte de comprensión de la base de datos, de las consultas SQL y del funcionamiento del servidor no es necesaria y se procedió directamente a la escritura de los programas necesarios.

En este caso solo hizo falta la escritura de un programa async_monitor.c y su respectivo archivo de cabecera .h debido a que al no haber nada parecido ni ninguna tabla asociada a dicho objeto se puede juntar tanto la definición del objeto como lo necesario para hacer la consulta (la parte request) en el mismo archivo.

En el archivo .h se definieron las variables que definen el objeto mediante una estructura. En este caso no se trata de las columnas de las tablas sino que la información que queremos trasladar desde la base de datos hasta la interfaz son el número de servidores, tareas y subtareas que hay en cada estado.

En el caso de los servidores los estados posibles son servidor en espera, trabajando y en error, para ello se realizan por lo tanto 3 variables processwaiting, processworking y processerror respectivamente.

En el caso de las tareas pueden estar inactivas, en espera, trabajando, terminadas, en error o canceladas. Se requieren por lo tanto las siguientes variables respectivamente, taskinactive, taskwaiting, taskworking, taskfinished, tasquero, taskaborted.

Las subtareas pueden estar en los mismos estados que las tareas añadiendo un estado suplementario, subtasksplit, en el que aparecerán el número de subtareas que han sido divididas.

```
1.  typedef struct __async_monitor__
2.  +{
3.  +    short typerequesti;
4.  +    short tagi;
5.  +
6.  +    //Processus
7.  +    long processwaiting;
8.  +    long processworking;
9.  +    long processerror;
10. +
11. +    //Task
12. +    long taskinactive;
13. +    long taskwaiting;
14. +    long tasksplit;
15. +    long taskworking;
16. +    long taskfinished;
17. +    long taskerror;
18. +    long taskaborted;
19. +
20. +    //Subtask
21. +    long subtaskinactive;
22. +    long subtaskwaiting;
23. +    long subtasksplit;
24. +    long subtaskworking;
25. +    long subtaskfinished;
26. +    long subtaskerror;
27. +    long subtaskaborted;
```

```
+
+} ASYNC_MONITOR;
```

El archivo .c tuvo que crearse desde 0 al contrario que en el caso de la función Display pero contiene las mismas funciones que las que hubo que añadir en el caso anterior. Primero la macro, la estructura en la que se definen las variables de forma que permita codificar la información para enviarla y descodificarla cuando se recibe. En este caso las variables también son externas a la tabla, es más fácil de verlo que en el caso anterior puesto que las acabamos de crear.

```
        M_INIT_DESC2(ASYNC_MONITOR, ASYNC_MONITOR_ID, "PATIO_PATIO", "ASYNC_MONITOR",
 MESS_TAG_ASYNC_MONITOR)
+
+    M_DATA_EXTR("TYPEREQUESTI",                SQL_SMALLINT,   SHORT,          t
yperequesti)
+    M_DATA_EXTR("TAGI",                        SQL_SMALLINT,   SHORT,          t
agi)
+
+    M_DATA_EXTR("PROCESSWAITING",              SQL_INTEGER,  LONG,          proces
swaiting)
+    M_DATA_EXTR("PROCESSWORKING",              SQL_INTEGER,  LONG,          proces
sworking)
+    M_DATA_EXTR("PROCESSERROR",                SQL_INTEGER,  LONG,          proces
serror)
+    M_DATA_EXTR("TASKINACTIVE",                SQL_INTEGER,  LONG,          taskin
active)
+    M_DATA_EXTR("TASKWAITING",                 SQL_INTEGER,  LONG,          taskwa
iting)
+    M_DATA_EXTR("TASKSPLIT",                   SQL_INTEGER,  LONG,          tasksp
lit)
+    M_DATA_EXTR("TASKWORKING",                 SQL_INTEGER,  LONG,          taskwo
rking)
+    M_DATA_EXTR("TASKFINISHED",                SQL_INTEGER,  LONG,          taskfi
nished)
+    M_DATA_EXTR("TASKERROR",                   SQL_INTEGER,  LONG,          tasker
ror)
+    M_DATA_EXTR("TASKABORTED",                 SQL_INTEGER,  LONG,          taskab
orted)
+    M_DATA_EXTR("SUBTASKINACTIVE",             SQL_INTEGER,  LONG,          subtas
kinactive)
+    M_DATA_EXTR("SUBTASKWAITING",              SQL_INTEGER,  LONG,          subtas
kwaiting)
+    M_DATA_EXTR("SUBTASKSPLIT",                SQL_INTEGER,  LONG,          subtas
ksplit)
+    M_DATA_EXTR("SUBTASKWORKING",              SQL_INTEGER,  LONG,          subtas
kworking)
+    M_DATA_EXTR("SUBTASKFINISHED",             SQL_INTEGER,  LONG,          subtas
kfinished)
+    M_DATA_EXTR("SUBTASKERROR",                SQL_INTEGER,  LONG,          subtas
kerror)
+    M_DATA_EXTR("SUBTASKABORTED",              SQL_INTEGER,  LONG,          subtas
kaborted)
+
+    M_OBJECTREQUEST3_FP(async_monitor_from_objectrequest);
+
+    M_END_DESC()
+    M_END_DESC_CLI()
+    M_TABLE_NOT_IN_DATABASE()
+}
```

Posteriormente se escriben las 3 funciones siguientes, ASYNC_MONITOR_FROM_OBJECTREQUEST, ASYNC_MONITOR_SERVER_PROCESS y ASYNC_MONITOR_SERVER_READ.

Las dos primeras son equivalentes a las realizadas en los casos anteriores, su objetivo es el de recuperar el tipo de consulta, que en este caso es de tipo lectura, asociarle un mess_tag y una identificación y llamar a la función SERVER_READ que va a realizar la acción.

En este caso la función de lectura es un poco diferente que en el caso anterior puesto que tiene que recuperar información de tres tablas diferentes y realizar unos contadores. En este caso la estructura de la consulta SQL será muy simple pues el usuario no introduce ningún parámetro. De forma genérica, se hará así, variando únicamente el nombre en función de si se trata de la tabla de tareas, servidores o subtareas,

SELECT STATUSI, COUNT(1) as nb_per_status
FROM “nombre de la tabla”
WHERE STATUSI >=0 GROUP BY STATUSI

La función COUNT se encarga de realizar un contador, suma uno cada vez que ve un estado igual, para que sea más comprensible se la ha puesto un título, nb_per_status aunque no es obligatorio. Por el momento se seleccionan únicamente los estados superiores a 0 a pesar de que para las subtareas y tareas existe el estado -1. Esto se debe a que la realización de esta parte del proyecto desveló un bug en las capas más bajas del servidor de la empresa que hasta ahora no había sido detectado pues en ningún momento se realizaba una consulta de este tipo. Hasta la finalización de este proyecto no se ha encontrado solución a este error por lo que por el momento sigue de este modo, una vez el error solucionado bastara con quitar “STATUSI >=0” para que el código funcione de forma óptima. Finalmente mediante el comando GROUP BY lo que se hace es implemente ordenar el resultado de menor a mayor.

Una vez que está hecha la consulta, con un simple switch case se asocia el número obtenido para cada estado a su variable correspondiente del objeto ASYNC_MONITOR. Al consultar la información en tres tablas diferentes es necesario realizar tres consultas y asociarle tres estructuras switch case tal y como se ve en el código siguiente donde la primera consulta se hace a la tabla de tareas, la segunda a la de subtareas y la última a la de servidores.

```
1.  strcpy (wheres, " WHERE STATUSI >= 0 GROUP BY STATUSI");
2.  +
3.  +    inventmonitor = node_list_init();
4.  +
5.  +    memset(&async_monitor, '\0', sizeof(async_monitor));
6.  +
7.  +    DBEXTREAD(ASYNC_TASK_ID, 0, "select STATUSI, COUNT(1) as nb_per_status", 0, whe
    res, NULL, 1, &statusi, 2, &nb_per_status, 0, NULL, 0, NULL, 0, NULL, 0, NULL, 0, NU
    LL, 0, 0, 0, 0, 0, 0)
8.  +    {
9.  +        switch (statusi)
10. +        {
11. +        case ASYNC_DEF_STATUS_INACTIVE:
12. +        async_monitor.taskinactive = nb_per_status;
13. +        break;
14. +
```

```
15. +         case ASYNC_DEF_STATUS_WAITING:
16. +         async_monitor.taskwaiting = nb_per_status;
17. +         break;
18. +
19. +         case ASYNC_DEF_STATUS_SPLIT:
20. +         async_monitor.tasksplit = nb_per_status;
21. +         break;
22. +
23. +         case ASYNC_DEF_STATUS_WORKING:
24. +         async_monitor.taskworking = nb_per_status;
25. +         break;
26. +
27. +         case ASYNC_DEF_STATUS_FINISHED:
28. +         async_monitor.taskfinished = nb_per_status;
29. +         break;
30. +
31. +         case ASYNC_DEF_STATUS_ERROR:
32. +         async_monitor.taskerror = nb_per_status;
33. +         break;
34. +
35. +         case ASYNC_DEF_STATUS_ABORTED:
36. +         async_monitor.taskaborted = nb_per_status;
37. +         break;
38. +
39. +         default : seraudit_var("Unknown status %d nb %ld ", statusi, nb_per_status)
    ;
40. +         }
41. +     }
42. +
43. +     DBEXTREAD(ASYNC_SUBTASK_ID, 0, "select STATUSI, COUNT(1) as nb_per_status", 0,
    wheres, NULL, 1, &statusi, 2, &nb_per_status, 0, NULL, 0, NULL, 0, NULL, 0, NULL, 0,
     NULL, 0, 0, 0, 0, 0, 0)
44. +     {
45. +         switch (statusi)
46. +         {
47. +         case ASYNC_DEF_STATUS_INACTIVE:
48. +         async_monitor.subtaskinactive = nb_per_status;
49. +         break;
50. +
51. +         case ASYNC_DEF_STATUS_WAITING:
52. +         async_monitor.subtaskwaiting = nb_per_status;
53. +         break;
54. +
55. +         case ASYNC_DEF_STATUS_SPLIT:
56. +         async_monitor.subtasksplit = nb_per_status;
57. +         break;
58. +
59. +         case ASYNC_DEF_STATUS_WORKING:
60. +         async_monitor.subtaskworking = nb_per_status;
61. +         break;
62. +
63. +         case ASYNC_DEF_STATUS_FINISHED:
64. +         async_monitor.subtaskfinished = nb_per_status;
65. +         break;
66. +
67. +         case ASYNC_DEF_STATUS_ERROR:
68. +         async_monitor.subtaskerror = nb_per_status;
69. +         break;
70. +
71. +         case ASYNC_DEF_STATUS_ABORTED:
72. +         async_monitor.subtaskaborted = nb_per_status;
73. +         break;
74. +
75. +         default : seraudit_var("Unknown status %d nb %ld ", statusi, nb_per_status)
    ;
76. +         }
```

```
77. +    }
78. +
79. +    DBEXTREAD(ASYNC_REGISTRATION_ID, 0, "select STATUSI, COUNT(1) as nb_per_status"
    , 0, wheres, NULL, 1, &statusi, 2, &nb_per_status, 0, NULL, 0, NULL, 0, NULL, 0, NUL
    L, 0, NULL, 0, 0, 0, 0, 0, 0)
80. +    {
81. +        switch (statusi)
82. +        {
83. +        case ASYNC_DEF_STATE_WAITING:
84. +        async_monitor.processwaiting = nb_per_status;
85. +        break;
86. +
87. +        case ASYNC_DEF_STATE_WORKING:
88. +        async_monitor.processworking = nb_per_status;
89. +        break;
90. +
91. +        case ASYNC_DEF_STATE_ZOMBIE:
92. +        async_monitor.processerror = nb_per_status;
93. +        break;
94. +
95. +        default : seraudit_var("Unknown status %d nb %ld ", statusi, nb_per_status)
    ;
96. +        }
97. +    }
```

DBEXTREAD es una macro ya existente creada para poder construir las consultas SQL a las bases de datos. Se usa cuando se hacen consultas de tipo lectura y generalmente cuando no son de tipo básico.

Una vez todas las variables rellenadas con su contador correspondiente se rellena el puntero que lo contendrá todo y se envía al cliente mediante la función object_server_invent_send_bundle de la forma siguiente:

```
1. +    node_list_add_elt(inventmonitor, &async_monitor, sizeof(async_monitor));
2. +    memset(&async_monitor, '\0', sizeof(async_monitor));
3. +
4. +    object_server_invent_send_bundle(ASYNC_MONITOR_ID, NULL, inventmonitor, socket)
    ;
5. +
6. +    node_list_dispose(inventmonitor);
```

Una vez programado el código con el que el servidor es capaz de enviar la información, es hora de programar la parte cliente, si bien es cierto que por el momento no ha sido posible comprobar el correcto funcionamiento de esta parte.

Se empieza con la construcción de los APIJAVA donde solo será necesaria la creación de una clase, ASYNC_MONITOR en la que se definen las variables del objeto, teniendo en cuenta que tengan el mismo nombre que del lado del servidor para que la comunicación a los dos lados del socket sea posible, y se inicializan con un valor no definido mediante una función que realiza una consulta vacía.

Posteriormente se realizan las dos funciones que permiten las consultas de modo equivalente al caso de la función Display. En este caso, al no tener que introducir ningún valor la primera función solo se introduce el valor del tipo de consulta. La segunda función se encarga de llamar a la función anterior, de construir la socket y de gestionar el almacenamiento del resultado recibido. Al final de la clase se encuentran como es habitual los get/set de cada parámetro.

```
1.  private static OBJECTREQUEST initAsync_MonitorRequest()
2.  +       {
3.  +               OBJECTREQUEST objectrequest = new OBJECTREQUEST(new ASYNC_MONITOR())
                    objectrequest.setTypeRequesti(SYSENV.TYPEREQUEST_READ);
4.  +
5.  +               return objectrequest;
6.  +       }
7.  +
8.  +
9.
10. public static List ReadFromServer(PatioSocket patiosocket) throws PatioException
11. +       {
12. +               OBJECTREQUEST objectrequest = initAsync_MonitorRequest();
13. +
14. +               ActioList list = objectrequest.getActioList(patiosocket);
15. +
16. +               List Result = new ArrayList();
17. +               for (int i = 0; i < list.getList().size(); i++)
18. +               {
19. +             Object elt = list.getList().get(i);
20. +             ASYNC_MONITOR newAsync_monitor = (ASYNC_MONITOR) elt;
21. +             Result.add(newAsync_monitor);
22. +               }
23. +
24. +               return Result;
```

También se ha incluido una función toString que no es necesaria para el funcionamiento del proyecto pero que permite realizar llegados a este punto la comprobación del funcionamiento de la parte servidor y su comunicación con el APIJAVA. Para ello se ha completado la clase de prueba que tiene creada la empresa añadiéndole la clase que acabamos de crear.

A continuación se muestra el diálogo obtenido cuando se quiere probar el funcionamiento a la hora de buscar, contar y mostrar los estados,

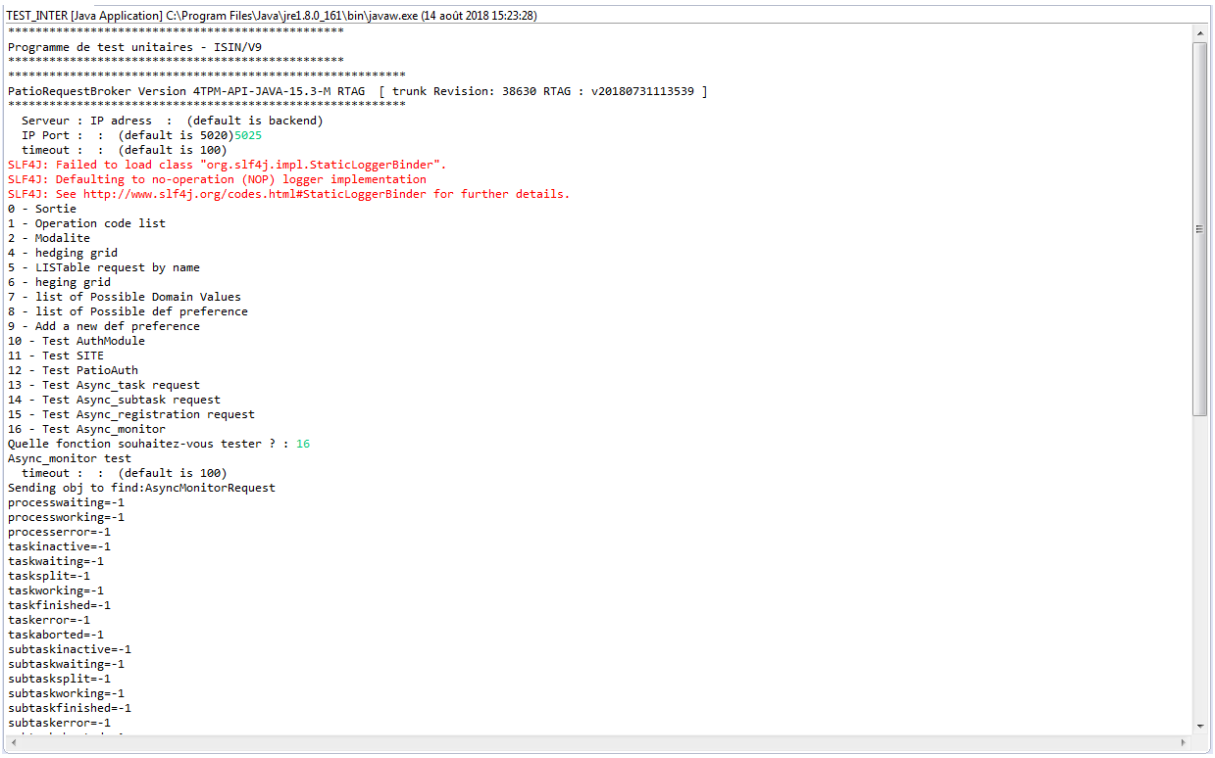


Figura 15: Diálogo de prueba en el caso de una consulta del estado de los servidores, tareas y subtareas asíncronos

En este caso se puede observar que son valores inusuales, lo que permite darse cuenta de que hay un error en alguna parte. Que el valor sea -1 es significativo, es el valor que se suele asignar en JAVA cuando el parámetro enviado o recibido esta vacío. Por lo tanto permite pensar que el error se halla en la comunicación entre el cliente y el servidor. Este test es por lo tanto muy útil para saber rápidamente, puesto que de toda la estructura del cliente solo se ha hecho la capa más baja, si el código funciona.

Una vez que se ha comprobado el correcto funcionamiento se procede a la programación de las capas superiores.

Esta parte requiere de más o menos las mismas clases que en la anterior función, adaptándolas a sus particularidades.

Primero se crean dos objetos, uno que representa el objeto de la funcionalidad, llamado AsyncMonitorObj, es el equivalente de lo que en la anterior función se llamó AsyncMonitorDisplay, y otro en el cual se definen las variables que componen la función, llamado simplemente AsyncMonitor, cuyo equivalente podría ser AsyncMonitorTask. En este caso en el que es tan simple y la función no requiere que se le introduzcan parámetros resulta un poco complicado de entender porque se requiere de dos clases diferentes pero son limitaciones del lenguaje que no permite juntar todo en una clase ya que, entre otras razones, cada una extiende clases diferentes.

Seguidamente se definen las funciones encargadas de abrir la pantalla y de crear las ventanas de parámetros y de resultados. En la primera clase simplemente se especifica el

nombre del editor y de la función, el resto son funciones genéricas que se encargan de gestionar la apertura de la ventana.

Posteriormente se realiza la del editor de la función, en este caso es el editor más básico posible puesto que no es necesario crear ventana de parámetros, ni consulta, simplemente se llama al servicio que se encarga de realizar la unión con las APIJAVA sin enviarle nada. Dicho servicio sigue siendo BlockOrderServiceImpl puesto que esta función está relacionada también con las órdenes en bloque. En ella simplemente se especifica que la consulta es de tipo lectura, sin pasar ningún parámetro y se hace la conversión entre el resultado que obtiene la APIJAVA proveniente del servidor hacia el objeto de tipo servicio web.

Como última función de este grupo hay que realizar la clase encargada de la construcción de la ventana de resultados. Llamada AsyncMonitorResult es una clase parecida a la anterior donde gráficamente se construye la apariencia gráfica del resultado. En este caso al mostrar datos relativos por un lado a los servidores y por otro de las tareas y subtareas se ha elegido separar la habitual tabla de resultados en dos, una para cada tipo. Para eso simplemente se crean dos "composite" gráficos tal y como se muestra a continuación.

```
1.  protected void buildResultContent(Composite parent, List<AsyncMonitor> data, FormToo
    lkit toolkit, SWTViewer swtViewer)
2.  +    {
3.  +        TableData tableDataprocess = new TableData(3198L, AsyncMonitorprocessDataSe
    t.class.getName(), "TableAsyncMonitorProcess",
4.  +                null);
5.  +        Composite compositeprocess = IhmUtilHolder.getIhmUtil().createComposite(par
    ent, SWT.NONE, null);
6.  +        compositeprocess.setLayout(new GridLayout());
7.  +        swtViewer.buildTableContent(compositeprocess, tableDataprocess);
8.  +
9.  +        TableData tableData = new TableData(3269L, AsyncMonitorDataSet.class.getNam
    e(),
10. +                "TableAsyncMonitor", null);
11. +        Composite composite = IhmUtilHolder.getIhmUtil().createComposite(parent, SW
    T.NONE, null);
12. +        composite.setLayout(new GridLayout());
13. +        swtViewer.buildTableContent(composite, tableData);
14. +    }
```

Al haber dos tablas de resultados diferentes se requiere de dos DataSet diferentes, uno para cada una. Es por eso que es necesario crear uno llamado AsyncMonitorDataSet, relativo a las tareas y a las subtareas. Y también uno llamado AsyncMonitorProcessDataSet centrado en los servidores. En ambos casos las clases contienen dos funciones, una donde se crean las columnas dándoles un título y especificando que tipo de dato van a contener y otra que se encarga de introducir dicho valor en su célula correspondiente. En el código siguiente se muestra dicha clase en el caso de los servidores que presenta la particularidad de que cuando el número de servidores en error es superior a 0 debe aparecer en color rojo.

```
1.      protected List<ReportColumnHolder> getColumns(String tableName)
2.  +    {
3.  +        reportColumnHolders
```

```
+                .add(new ReportColumnHolder("processwaiting", "async.monitor.processwaiting", TPMColumnType.INTEGER, true, true));
+        reportColumnHolders
+                .add(new ReportColumnHolder("processworking", "async.monitor.processworking", TPMColumnType.INTEGER, true, true));
+
+        ReportColumnHolder reportColumnHolder = new ReportColumnHolder("processerror", "async.monitor.processerror",
+                TPMColumnType.INTEGER, true, true);
+        List<TPMHighlightRule> highlightRules = new ArrayList<TPMHighlightRule>();

+        Map<String, String> properties = new HashMap<String, String>();
+        properties.put(HighlightRule.COLOR_MEMBER, ColorUtil.COLOR_CODE_RED);
+        TPMHighlightRule tpmHighlightRule = new TPMHighlightRule("row[\"processerror\"]", DesignChoiceConstants.MAP_OPERATOR_GT,
+                "0", null, properties);
+        highlightRules.add(tpmHighlightRule);
+        reportColumnHolder.setHighlightRule(highlightRules);
+        reportColumnHolders
+                .add(reportColumnHolder);
+
+        return reportColumnHolders;
+    }
+
+    @Override
+    protected void performFetch(IUpdatableDataSetRow row, AsyncMonitor item) throws ScriptException
+    {
+        setRowColumnValue(row, "processwaiting", item.getProcesswaiting(), item.getClass().getSimpleName());
+        setRowColumnValue(row, "processworking", item.getProcessworking(), item.getClass().getSimpleName());
+        setRowColumnValue(row, "processerror", item.getProcesserror(), item.getClass().getSimpleName());
+    }
```

Aunque existan dos tablas de resultados diferentes solo es necesario una clase de rptdesign para realizar la exportación a otros formatos puesto que Eclipse da la posibilidad de añadir dos clases DataSet a una misma clase rptdesign lo que resulta muy práctico para poder tener los dos resultados juntos, que es lo que se desea.

Finalmente es necesaria una clase que sirva de adaptador entre las variables presentes en la APIJAVA y las de las capas superiores que se ha llamado simplemente AsynMonitorAdapter. Es una clase muy similar al ejemplo que se ha mostrado anteriormente donde se introduce dentro de la variable de tipo PATIO el contenido de la variable de tipo APIJAVA.

Finalmente una vez que se han creado estas clases hay que realizar las extensiones para que la funcionalidad sea visible en la plataforma y funcione correctamente. Para ello se procede de forma similar que en el caso de la función anterior, mediante un plug-in utilizando o bien la interfaz gráfica o directamente escribiendo el código en XML en los casos en los que resulte más sencillo.

En este caso también es necesario añadir extensiones en la categoría definición, editor y función, el contexto será el mismo que anteriormente por lo que no hay que añadir nada. La mecánica en el caso de la definición y del editor no tiene nada de particular y se procede de

la misma forma que antes, añadiendo los nombres, códigos de identificación y clases propias de esta función.

El caso de la categoría función es interesante puesto que los contextos en que se quiere que la funcionalidad sea visible y disponible cambian. Se quiere que la función esté disponible siempre que sea visible por lo que se introducen las mismas condiciones en ambos casos.

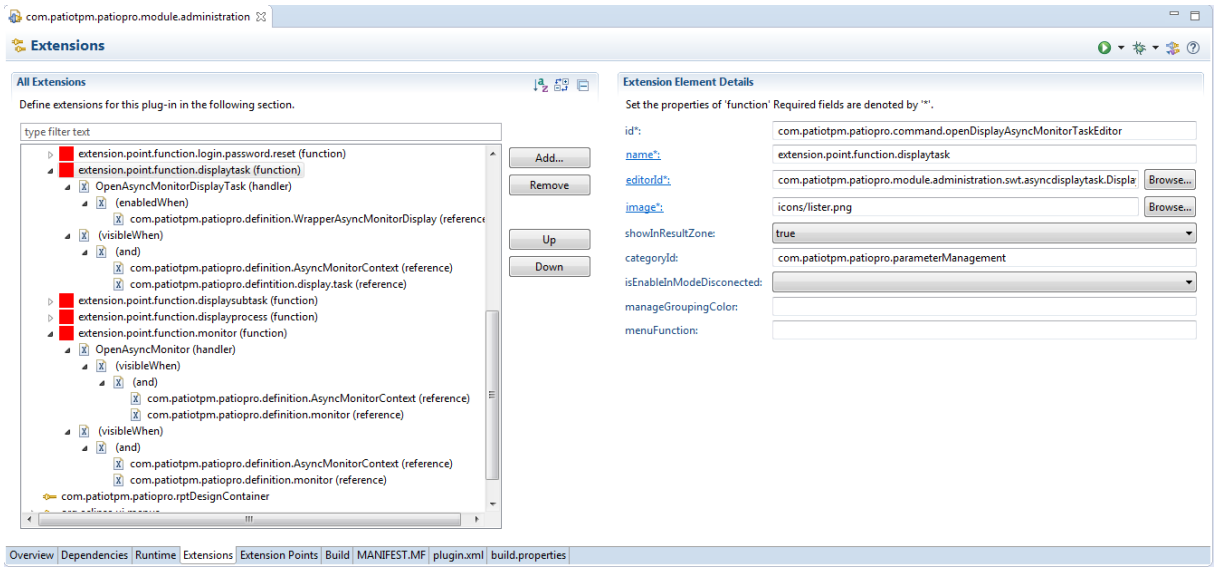


Figura 16: Categoría de funciones extendida del plug-in de extensiones (bis)

En seguida se muestra el código XML de esta extensión puesto que es más fácil de entender de este modo,

```
      <function
            categoryId="com.patiotpm.patiopro.parameterManagement"

editorId="com.patiotpm.patiopro.module.administration.swt.asyncmonitor.AsyncMonito
rEditor"
            id="com.patiotpm.patiopro.command.openAsyncMonitorEditor"
            image="icons/lister.png"
            name="extension.point.function.monitor"
            showInResultZone="true">
         <handler

class="com.patiotpm.patiopro.module.administration.swt.asyncmonitor.OpenAsyncMonit
or">
          <visibleWhen>
            <and>
                <reference

definitionId="com.patiotpm.patiopro.definition.AsyncMonitorContext">
                </reference>
             <reference
                   definitionId="com.patiotpm.patiopro.definition.monitor">
             </reference>
            </and>
         </visibleWhen>
         </handler>
         <visibleWhen>
```

```
            <and>
               <reference
definitionId="com.patiotpm.patiopro.definition.AsyncMonitorContext">
               </reference>
               <reference
                    definitionId="com.patiotpm.patiopro.definition.monitor">
               </reference>
            </and>
         </visibleWhen>
      </function>
```

Por último, además de las clases que se acaban de comentar también se han creado ciertas clases que aunque son necesarias para el correcto funcionamiento de la función, no son de especial interés puesto que se trata de funciones genéricas que hay que añadir para todas las funcionalidades y sin ninguna particularidad de interés.

Una vez hecho esto las dos funciones, tanto Display como Monitor están operativas siguiendo con las especificaciones iniciales.

4.3 Mejoras del proyecto inicial

Llegados a este punto se completó con éxito el proyecto siguiendo con las especificaciones iniciales. Una vez la base hecha y operativa hubo tiempo para pensar en posibles mejoras siguiendo con la dinámica inicial, principalmente pensando en facilitar la búsqueda que realizará el equipo de soporte.

Desde el principio se pensó que sería útil, aunque no una prioridad y por eso no se puso en las especificaciones iniciales, que desde el equipo de soporte se puedan realizar búsquedas introduciendo el tiempo en segundos del último heartbeat de los servidores. El resultado muestra en este caso todos los servidores que llevan más de ese tiempo en segundos sin mandar una señal de que están trabajando. Esto resulta interesante puesto que es posible que un servidor que no ha actualizado dicho valor, aunque su estado sea el de operativo, este realmente en error. Ser capaces de buscar este desfase es especialmente práctico a la hora de realizar un seguimiento más preciso y en tiempo real de los servidores.

Esta mejora se añadió en la ventana de parámetros de la categoría process y permite la búsqueda introduciendo un valor en segundos tal y como se aprecia en la captura siguiente que es una vista detallada de la figura número 23 que muestra los parámetros de búsqueda de servidores.

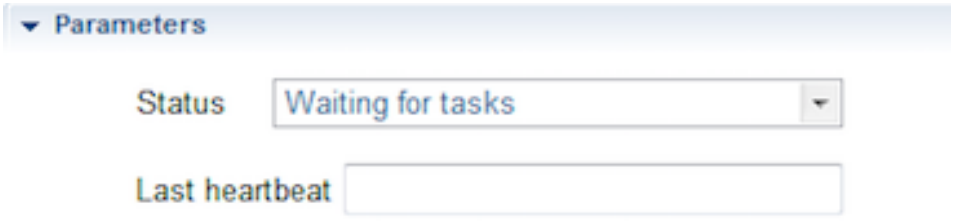


Figura 17: Zoom de la ventana de parámetros de la función Display para un servidor

Para la realización de esta mejora simplemente se ha realizado el mismo procedimiento que para las otras mejoras, es decir añadir gráficamente la oportunidad de que el usuario introduzca el valor, esto se hace mediante la clase editor de la categoría process, es decir DisplayAsyncMonitorProcessEditor. A continuación se actualiza lo necesario para permitir realizar la consulta con este nuevo parámetro en la parte cliente, en los APIJAVA y finalmente se actualiza en el servidor añadiendo un if, de modo que si el servidor detecta que la variable no está vacía, añade dicho valor en la consulta SQL.

La segunda mejora que se ha implantado es, igual que anteriormente, añadir un parámetro de búsqueda, en este caso para las subtareas. Este parámetro permite buscar subtareas en función del código de identificación de una tarea. De este modo si después de haber estado mirando la información de una tarea, queremos analizar la información sobre las subtareas que se han creado a partir de ella, sea la razón que sea por ejemplo si conocemos dicho código de una tarea que ha fallado, podemos realizar rápidamente la búsqueda de todas las subtareas de dicha tarea a través de su código de identificación

Se puede utilizar esta mejora desde la ventana de parámetros de la categoría subtask, donde simplemente hay que introducir el código de identificación que se desee. En seguida se muestra un zoom de la figura número 26 que muestra los parámetros de búsqueda de las subtareas.

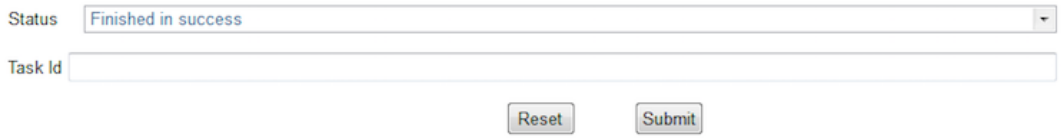


Figura 18: Zoom de la ventana de parámetros de la función Display para una subtarea

Para realizar esta mejora se procede igual que en el caso de la mejora anterior, es decir del mismo modo que para cualquier otro parámetro.

Una de las mejoras más interesantes que se implementó pertenece a la categoría de las tareas. Está relacionada con la mejora anterior pues pretende reflejar de manera más práctica la relación que hay por definición entre las tareas y las subtareas. Consiste en que una vez que se ha realizado la búsqueda y se ha desplegado la tabla de resultados que muestra las tareas requeridas, al pulsar dos veces sobre cualquiera de ellas se abre en la parte inferior de la pantalla una vista en detalle. En dicha vista se obtienen todas las subtareas en las que se divide la tarea, con toda la información relativa a ellas tal y como se obtendría realizando una búsqueda a partir de la categoría de subtareas.

Se trata de una funcionalidad muy práctica puesto que simplemente con un doble clic se dispone de toda la información sobre las subtareas en las que se divide la tarea, lo que evita tener que memorizar el código de identificación de la tarea, o cualquiera de sus características, ir a la categoría de subtareas y buscar las subtareas.

A continuación se muestra un ejemplo de una vista en detalle, se trata de un zoom de la figura 32,

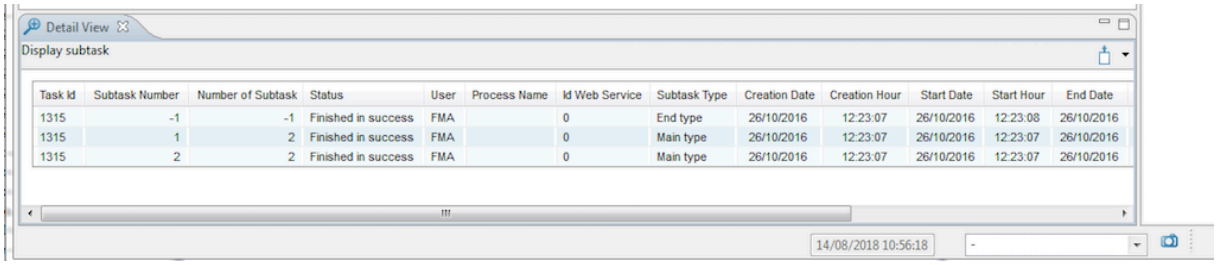


Figura 19: Vista en detalle de una tarea asíncrona

Esta mejora parece la más compleja de realizar pero sin embargo no es tan complicada. Para ello solo es necesario añadir ciertas funciones en la clase que gestiona el resultado para las tareas, DisplayAsyncMonitorTaskResult. Dichas funciones se encargan de abrir la ventana de la vista en detalle, de realizar la consulta SQL con los parámetros de la tarea en la cual se ha hecho el doble clic. Particularmente el parámetro que se ha escogido es su código de identificación lo que permite, gracias a la mejora anterior, obtener fácilmente sus subtareas. Finalmente la función se encarga de rellenar la vista en detalle con el resultado, en este caso esto es sencillo puesto que es exactamente la misma tabla que para las subtareas con lo cual simplemente se ha reutilizado dicha tabla.

A continuación se muestran dichas funciones,

```
1. +    protected boolean isWithDetail()
2. +    {
3. +        return true;
4. +    }
5. +
6. +
7. +    @Override
8. +    public void addExtraData(ReportRowHolder row)
9. +    {
10. +       if (row.getData() != null)
11. +       {
12. +           IWorkbenchPage page = PlatformUI.getWorkbench().getActiveWorkbenchWindo
    w().getActivePage();
13. +           DetailView detailView = (DetailView) page.findView(DetailView.ID);
14. +           detailView.setContentDescription(I18nUtil.getI18nMessage("display.async
    monitorsubtask.editor.title"));
15. +
16. +           AsyncMonitorTask TaskInput = (AsyncMonitorTask) row.getData();
17. +           AsyncMonitorSubtaskRequest SubtaskRequest = new AsyncMonitorSubtaskRequ
    est();
18. +           SubtaskRequest.setid_taskl(TaskInput.getId_taskl());
19. +           SubtaskRequest.setStatus(AsyncMonitorTaskStatus.ALL);
20. +
21. +           try
22. +           {
23. +               asyncMonitorSubtasks = ServiceFactory.getBlockOrderService()
24. +                       .asyncMonitorSubtask(UIUserContext.getCurrent().getUserInfo
    (), SubtaskRequest);
25. +           }
26. +           catch (TPMServiceException e)
27. +           {
28. +               LOGGER.error("Failed to find one async monitor subtask", e);
29. +           }
```

```
30. +
31. +            row.setDetailViewData((Object) asyncMonitorSubtasks);
32. +
33. +        }
34. +    }
```

Inicialmente se pensó que la función Monitor solo mostraría el número de servidores, tareas o subtareas que estuviesen en error, trabajando, en espera, terminados o cancelados cuando sin embargo existen más estados posibles. La idea era poner únicamente lo esencial ya que se trata de ver rápidamente si hay un fallo. Sin embargo luego se pensó que puesto que el objetivo es encontrar fallos cuanta más información esté disponible mejor y que no tenía mucho sentido esconder ciertos datos por lo que finalmente se muestran todos los estados en los que están tanto los servidores como las tareas y subtareas. Más que una mejora es un cambio de planteamiento y fue muy fácil de implementar puesto que la información ya era enviada desde el servidor hasta el cliente, simplemente antes no se mostraba en la tabla final y ahora sí.

4.4 Complementos

Al tratarse de una interfaz gráfica de uso profesional hay multitud de pequeños detalles gráficos que hay que implementar independientemente de la funcionalidad que sea. Por ejemplo cuando se pasa el cursor por encima de cualquier botón o título aparece un subtítulo. Algunos de estos detalles, como ocurre con los subtítulos o con algunos detalles más se incluyen de manera automática recogiendo el título de la función por ejemplo, sin embargo la mayoría de estos detalles gráficos no son automáticos, por lo que forma parte del trabajo del desarrollador hacerlo. También puede ocurrir que se quiera cambiar la información que viene automáticamente, en dicho caso se procede de la misma forma y se necesitan funciones que sobrescriban el valor que era asignado por defecto.

Evidentemente todo este trabajo se realizó al final, una vez que el proyecto estaba terminado y funcionaba correctamente, y que hubo un poco de tiempo para centrarse en pulir los detalles más gráficos que aunque no son una prioridad sí que son importantes.

Otro elemento importante que incluye todas las funcionalidades deriva del hecho de que la plataforma PATIO esta disponible en varios idiomas. Esto implica que los nombres de todos los elementos que se van a mostrar visualmente en la pantalla no pueden estar directamente escritos “a fuego” (hard-code) en las diferentes clases, sino que es necesario utilizar un código al que luego se le va a asignar un nombre real en terceros archivos. Hay uno de estos archivos para cada idioma. Es lo que denominan internacionalización y es una etapa importante que no se debe olvidar puesto que provocaría la aparición de una palabra francesa en la versión en inglés o viceversa.

4.5 Recursos utilizados

Para desarrollar la parte servidor se utilizó lenguaje C y el programa Microsoft Visual Studio en el que se implementó la solución Patio, que contiene todos los programas que componen el servidor de la empresa. Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows que soporta diferentes lenguajes de

programación, en nuestro caso C. Permite a los desarrolladores crear aplicaciones que comuniquen entre estaciones de trabajo, páginas web, consolas y otros.

Para manipular la base de datos se utilizó Microsoft SQL Server, se trata de un sistema de gestión de bases de datos, SGBD en SQL desarrollado por Microsoft.

En cuanto a la parte cliente se desarrolló usando lenguaje JAVA y el programa Eclipse, en este caso también hubo que recuperar todo el entorno existente para el correcto desarrollo de los programas. Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma. Se usa generalmente para desarrollar entornos de desarrollo integrados como el IDE de Java, Java Development Toolkit.

También fue necesario el uso de XML, a través de Eclipse, en ciertos momentos durante la realización de la interfaz.

Se utilizó también JIRA, una herramienta en línea cuyo objetivo es el de gestionar las tareas de un proyecto y realizar el seguimiento de bugs e incidentes.

En la empresa disponen también de un programa de gestión de versiones llamado TortoiseSVN donde los desarrolladores actualizan la versión de referencia con las modificaciones que han realizado localmente en sus ordenadores. Gracias a este programa es sencillo comprobar todas las modificaciones y programas que se han creado comparando la versión local que se encuentra en el ordenador con la versión de referencia que se encuentra en el servidor.

La integralidad del proyecto se realizó en las oficinas de la empresa con acceso a los softwares necesarios y a toda su estructura.

5. Resultados y Análisis

A continuación se muestra el resultado final tal y como se entregó al final de las prácticas, incluyendo por lo tanto no solo las especificaciones iniciales pero también las mejoras incluidas posteriormente.

Todas las imágenes que aparecen a continuación son capturas de pantalla de la plataforma PATIO, en este caso se trata de un PATIO ejecutado en un servidor local pero el resultado será idéntico una vez que se ejecute en un entorno real de producción.

5.1 Página principal

La primera captura muestra la pantalla inicial de la categoría Administración, donde se puede apreciar donde se ha implementado la funcionalidad Async Monitor, es decir en la categoría llamada “Scorecard” (ficha de evaluación en español).

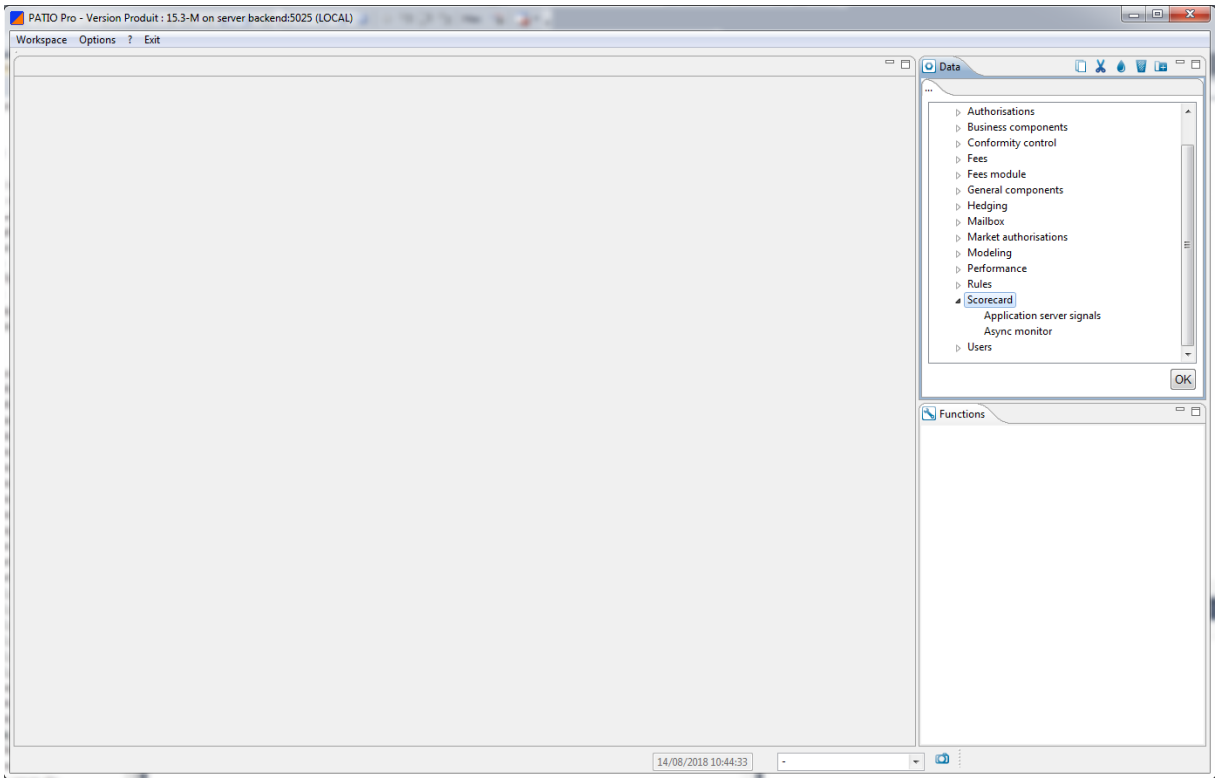


Figura 20: Página principal de PATIO

Una vez que se pulsa en Async Monitor se abre la ventana siguiente:

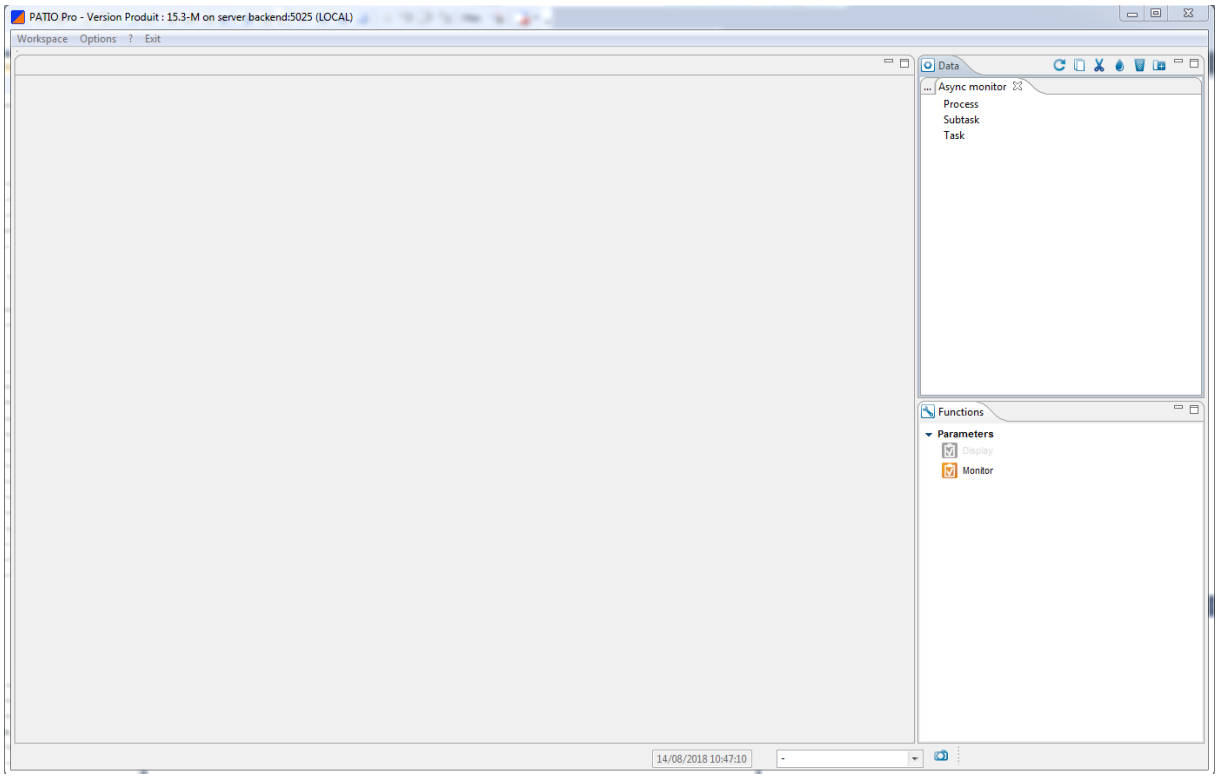


Figura 21: Página principal de la categoría de Async Monitor

Se puede apreciar que al no estar seleccionado ninguno de los datos, process, task o subtask la función Display no está disponible, aparece en gris. Siguiendo con las

especificaciones una vez que se selecciona alguno de los datos la función ya sí que está disponible tal y como se muestra a continuación.

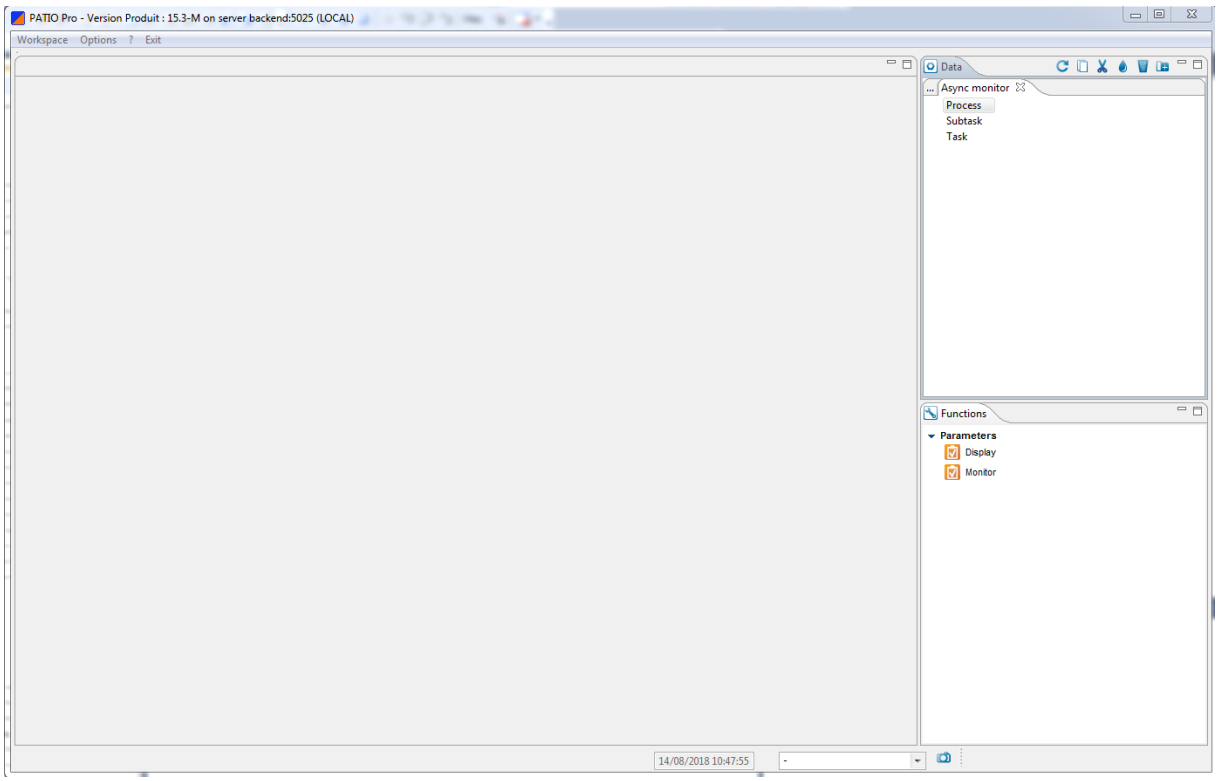


Figura 22: Página principal de la categoría de Async Monitor, dato seleccionado

5.2 Servidores

A continuación se muestra la funcionalidad para el caso de una búsqueda de los servidores utilizando la función Display. La primera captura corresponde a la ventana de parámetros en la que el usuario introduce el valor de las variables con las que desea realizar la búsqueda. En este caso puede buscar los servidores en función de su estado y también los servidores que lleven más de x tiempo sin dar señales de vida. La segunda captura muestra el resultado, la tabla que se obtiene al realizar la búsqueda más básica, es decir todos los servidores. Finalmente se muestra una captura del documento en PDF que se obtiene cuando se exporta el resultado a dicho formato.

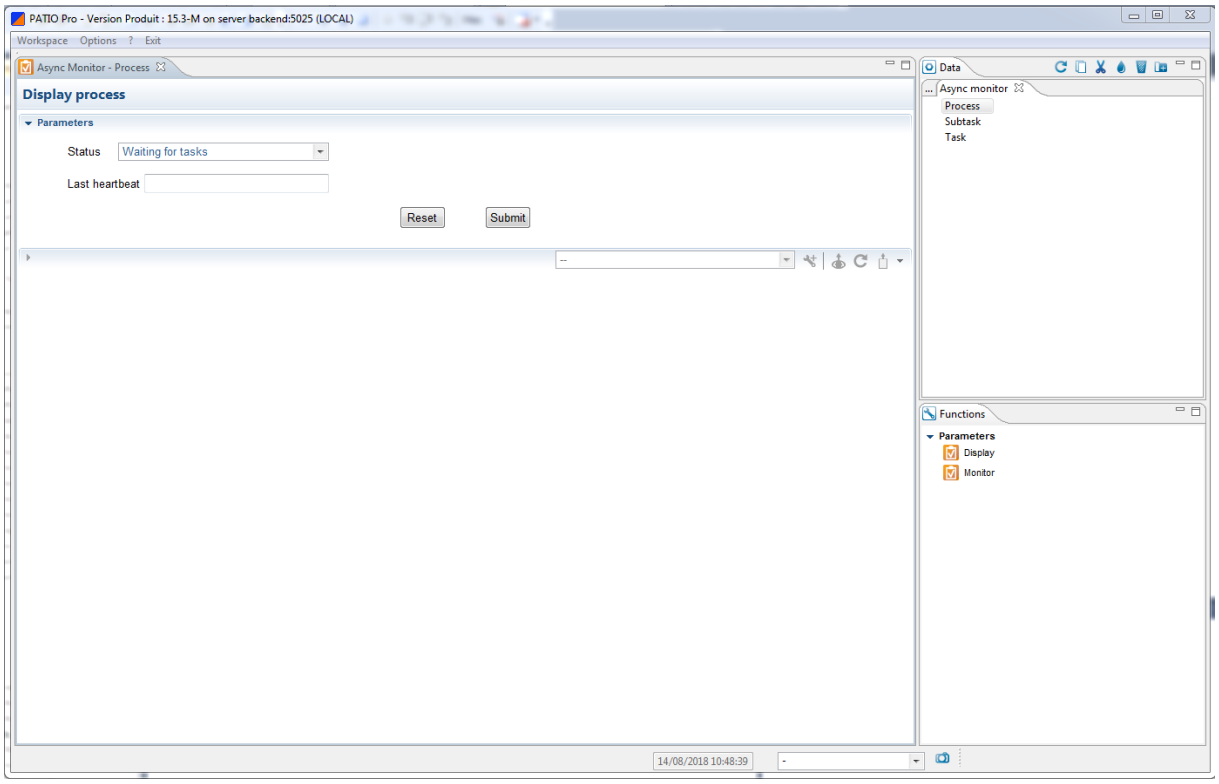


Figura 23: Ventana de parámetros de la función Display para un servidor

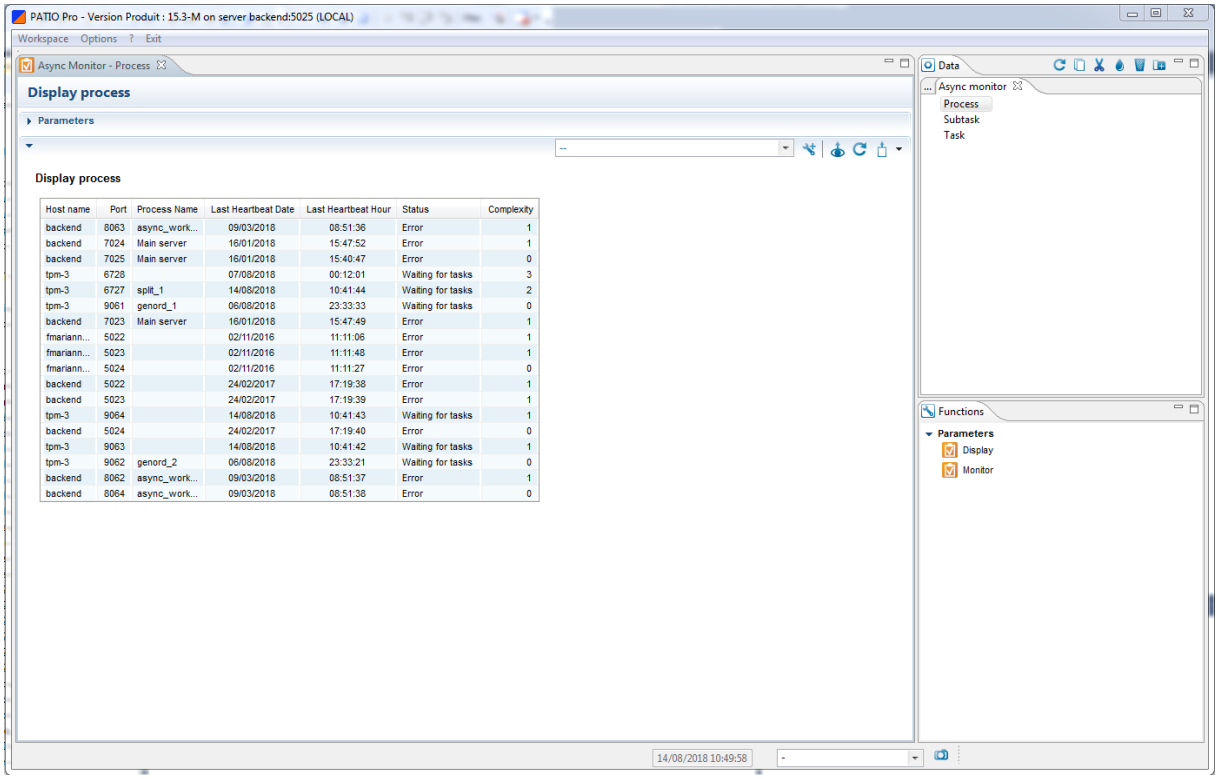


Figura 24: Resultado de una consulta de búsqueda de servidores

Host name	Port	Process Name	Last Heartbeat Date	Last Heartbeat Hour	Status	Complexity
backend	8063	async_work_2	09/03/2018	08:51:36	Error	1
backend	7024	Main server	16/01/2018	15:47:52	Error	1
backend	7025	Main server	16/01/2018	15:40:47	Error	0
tpm-3	6728		07/08/2018	00:12:01	Waiting for tasks	3
tpm-3	6727	split_1	14/08/2018	10:41:44	Waiting for tasks	2
tpm-3	9061	genord_1	06/08/2018	23:33:33	Waiting for tasks	0
backend	7023	Main server	16/01/2018	15:47:49	Error	1
fmarianne-pc	5022		02/11/2016	11:11:06	Error	1
fmarianne-pc	5023		02/11/2016	11:11:48	Error	1
fmarianne-pc	5024		02/11/2016	11:11:27	Error	0
backend	5022		24/02/2017	17:19:38	Error	1
backend	5023		24/02/2017	17:19:39	Error	1
tpm-3	9064		14/08/2018	10:41:43	Waiting for tasks	1
backend	5024		24/02/2017	17:19:40	Error	0
tpm-3	9063		14/08/2018	10:41:42	Waiting for tasks	1
tpm-3	9062	genord_2	06/08/2018	23:33:21	Waiting for tasks	0
backend	8062	async_work_1	09/03/2018	08:51:37	Error	1
backend	8064	async_work_3	09/03/2018	08:51:38	Error	0

Figura 25: Captura del documento PDF generado al exportar el resultado de la búsqueda de un servidor

5.3 Subtareas

Del mismo modo, a continuación se muestra la funcionalidad para el caso de una búsqueda de las subtareas utilizando la función Display. La primera captura corresponde a la ventana de parámetros en la que el usuario introduce el valor de las variables con las que desea realizar la búsqueda. En este caso puede buscar las subtareas en función de un gran número de variables, su estado, el usuario que llevó a cabo la tarea, la fecha de creación de la subtarea al igual que un intervalo de tiempo dentro del cual la subtarea empezó y finalmente el código de identificación de la tarea a partir de la cual se crea la subtarea. La segunda captura muestra el resultado, la tabla que se obtiene al realizar la búsqueda más básica, es decir una fecha cualquiera y todos los estados. Finalmente se muestra una captura del documento en PDF que se obtiene cuando se exporta el resultado a dicho formato.

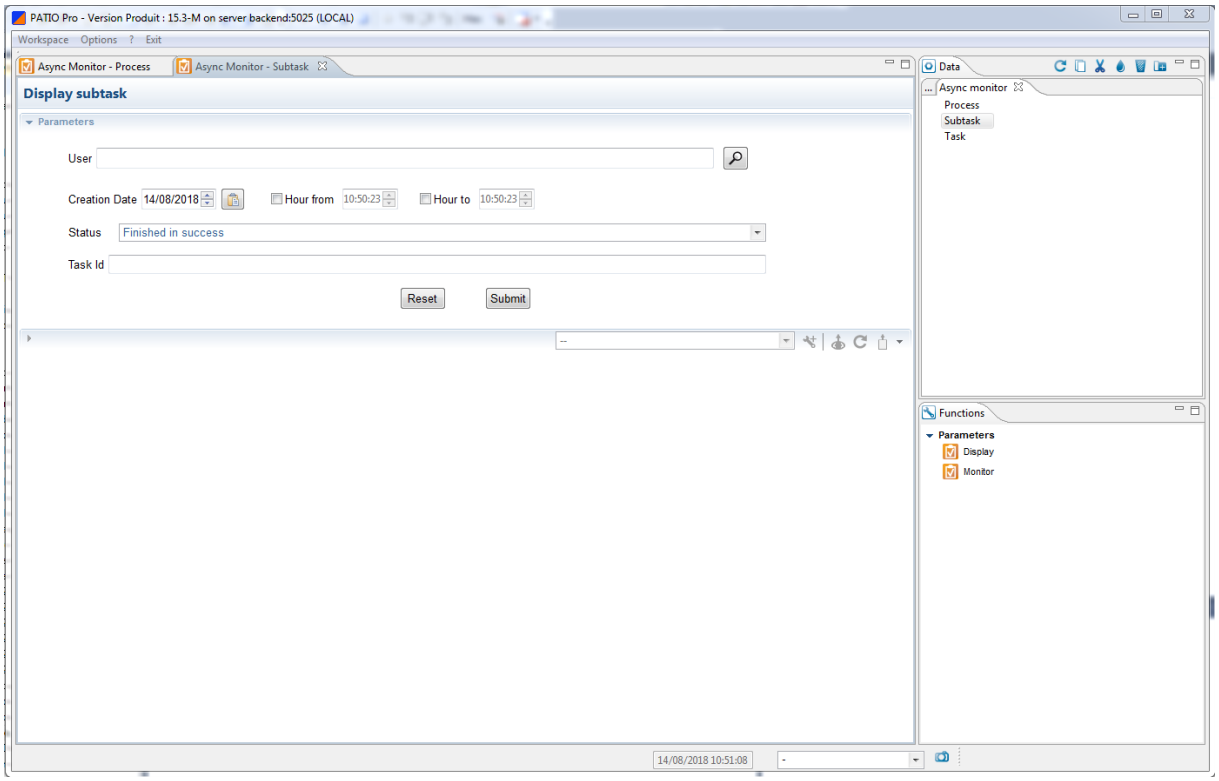


Figura 26: Ventana de parámetros de la función Display para una subtarea

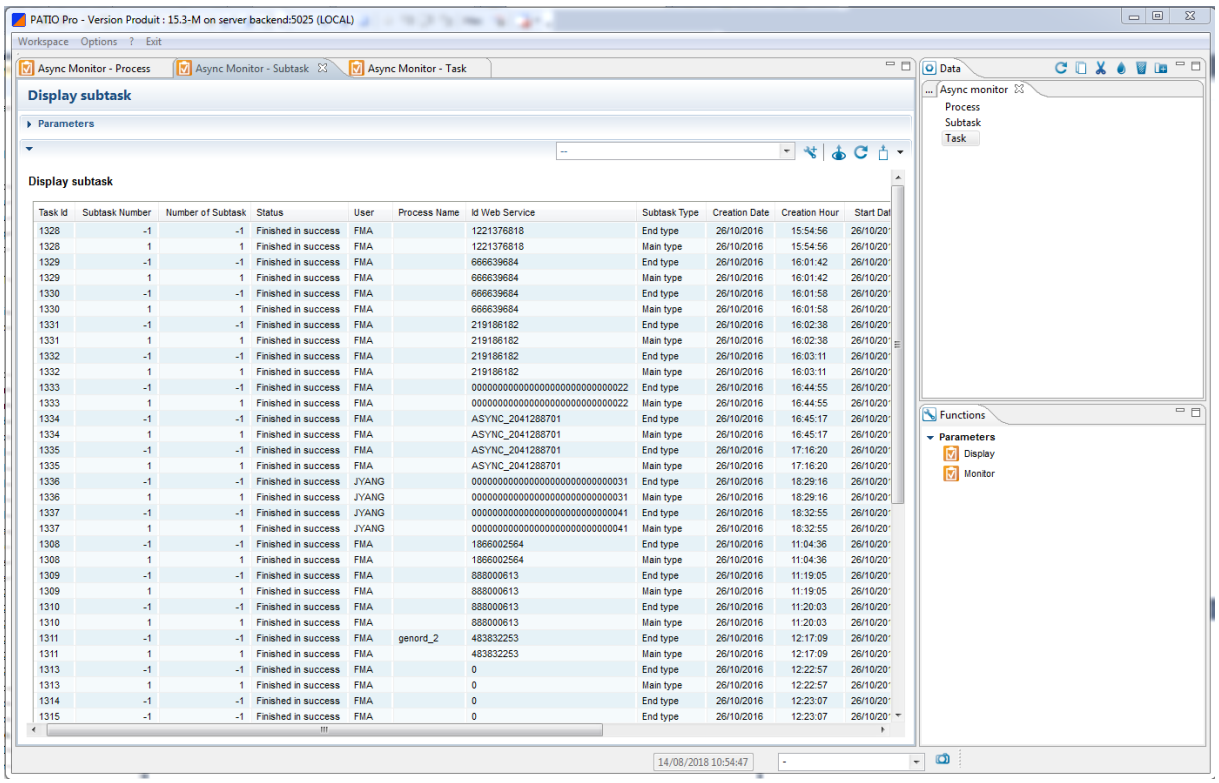
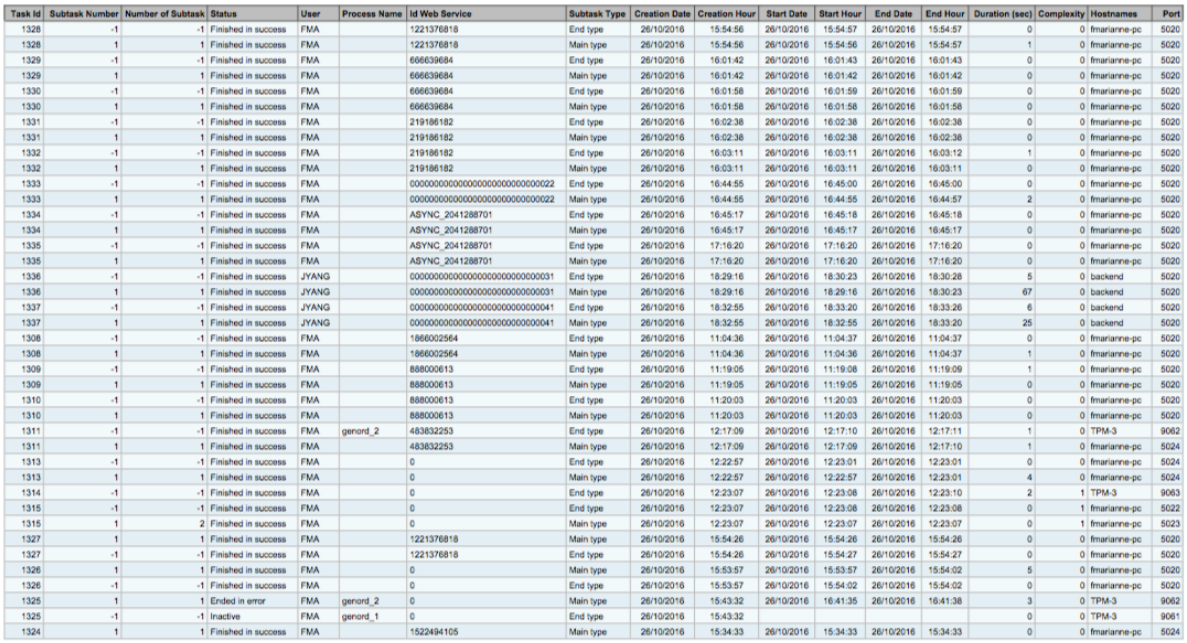


Figura 27: Resultado de la búsqueda de una subtarea



Task Id	Subtask Number	Number of Subtask	Status	User	Process Name	Id Web Service	Subtask Type	Creation Date	Creation Hour	Start Date	Start Hour	End Date	End Hour	Duration (sec)	Complexity	Hostnames	Port
1328	-1	-1	Finished in success	FMA		1221376818	End type	26/10/2016	15:54:56	26/10/2016	15:54:57	26/10/2016	15:54:57	0	0	fmarianne-pc	5020
1328	1	1	Finished in success	FMA		1221376818	Main type	26/10/2016	15:54:56	26/10/2016	15:54:56	26/10/2016	15:54:57	1	0	fmarianne-pc	5020
1329	-1	-1	Finished in success	FMA		666639684	End type	26/10/2016	16:01:42	26/10/2016	16:01:43	26/10/2016	16:01:43	0	0	fmarianne-pc	5020
1329	1	1	Finished in success	FMA		666639684	Main type	26/10/2016	16:01:42	26/10/2016	16:01:42	26/10/2016	16:01:42	0	0	fmarianne-pc	5020
1330	-1	-1	Finished in success	FMA		666639684	End type	26/10/2016	16:01:58	26/10/2016	16:01:59	26/10/2016	16:01:59	0	0	fmarianne-pc	5020
1330	1	1	Finished in success	FMA		666639684	Main type	26/10/2016	16:01:58	26/10/2016	16:01:58	26/10/2016	16:01:58	0	0	fmarianne-pc	5020
1331	-1	-1	Finished in success	FMA		219186182	End type	26/10/2016	16:02:38	26/10/2016	16:02:38	26/10/2016	16:02:38	0	0	fmarianne-pc	5020
1331	1	1	Finished in success	FMA		219186182	Main type	26/10/2016	16:02:38	26/10/2016	16:02:38	26/10/2016	16:02:38	0	0	fmarianne-pc	5020
1332	-1	-1	Finished in success	FMA		219186182	End type	26/10/2016	16:03:11	26/10/2016	16:03:11	26/10/2016	16:03:12	1	0	fmarianne-pc	5020
1332	1	1	Finished in success	FMA		219186182	Main type	26/10/2016	16:03:11	26/10/2016	16:03:11	26/10/2016	16:03:11	0	0	fmarianne-pc	5020
1333	-1	-1	Finished in success	FMA		00000000000000000000000000000022	End type	26/10/2016	16:44:55	26/10/2016	16:45:00	26/10/2016	16:45:00	0	0	fmarianne-pc	5020
1333	1	1	Finished in success	FMA		00000000000000000000000000000022	Main type	26/10/2016	16:44:55	26/10/2016	16:44:55	26/10/2016	16:44:57	2	0	fmarianne-pc	5020
1334	-1	-1	Finished in success	FMA		ASYNC_2041288701	End type	26/10/2016	16:45:17	26/10/2016	16:45:18	26/10/2016	16:45:18	0	0	fmarianne-pc	5020
1334	1	1	Finished in success	FMA		ASYNC_2041288701	Main type	26/10/2016	16:45:17	26/10/2016	16:45:17	26/10/2016	16:45:17	0	0	fmarianne-pc	5020
1335	-1	-1	Finished in success	FMA		ASYNC_2041288701	End type	26/10/2016	17:16:20	26/10/2016	17:16:20	26/10/2016	17:16:20	0	0	fmarianne-pc	5020
1335	1	1	Finished in success	FMA		ASYNC_2041288701	Main type	26/10/2016	17:16:20	26/10/2016	17:16:20	26/10/2016	17:16:20	0	0	fmarianne-pc	5020
1336	-1	-1	Finished in success	JYANG		00000000000000000000000000000031	End type	26/10/2016	18:29:16	26/10/2016	18:30:23	26/10/2016	18:30:28	5	0	backend	5020
1336	1	1	Finished in success	JYANG		00000000000000000000000000000031	Main type	26/10/2016	18:29:16	26/10/2016	18:29:16	26/10/2016	18:30:23	67	0	backend	5020
1337	-1	-1	Finished in success	JYANG		00000000000000000000000000000041	End type	26/10/2016	18:32:55	26/10/2016	18:33:20	26/10/2016	18:33:26	6	0	backend	5020
1337	1	1	Finished in success	JYANG		00000000000000000000000000000041	Main type	26/10/2016	18:32:55	26/10/2016	18:32:55	26/10/2016	18:33:20	25	0	backend	5020
1308	-1	-1	Finished in success	FMA		1866002564	End type	26/10/2016	11:04:36	26/10/2016	11:04:37	26/10/2016	11:04:37	0	0	fmarianne-pc	5020
1308	1	1	Finished in success	FMA		1866002564	Main type	26/10/2016	11:04:36	26/10/2016	11:04:36	26/10/2016	11:04:37	1	0	fmarianne-pc	5020
1309	-1	-1	Finished in success	FMA		888000613	End type	26/10/2016	11:19:05	26/10/2016	11:19:08	26/10/2016	11:19:09	1	0	fmarianne-pc	5020
1309	1	1	Finished in success	FMA		888000613	Main type	26/10/2016	11:19:05	26/10/2016	11:19:05	26/10/2016	11:19:05	0	0	fmarianne-pc	5020
1310	-1	-1	Finished in success	FMA		888000613	End type	26/10/2016	11:20:03	26/10/2016	11:20:03	26/10/2016	11:20:03	0	0	fmarianne-pc	5020
1310	1	1	Finished in success	FMA		888000613	Main type	26/10/2016	11:20:03	26/10/2016	11:20:03	26/10/2016	11:20:03	0	0	fmarianne-pc	5020
1311	-1	-1	Finished in success	FMA	genord_2	483832253	End type	26/10/2016	12:17:09	26/10/2016	12:17:10	26/10/2016	12:17:11	1	0	TPM-3	9062
1311	1	1	Finished in success	FMA		483832253	Main type	26/10/2016	12:17:09	26/10/2016	12:17:09	26/10/2016	12:17:10	1	0	fmarianne-pc	5024
1313	-1	-1	Finished in success	FMA		0	End type	26/10/2016	12:22:57	26/10/2016	12:23:01	26/10/2016	12:23:01	0	0	fmarianne-pc	5024
1313	1	1	Finished in success	FMA		0	Main type	26/10/2016	12:22:57	26/10/2016	12:22:57	26/10/2016	12:23:01	4	0	fmarianne-pc	5024
1314	-1	-1	Finished in success	FMA		0	End type	26/10/2016	12:23:07	26/10/2016	12:23:08	26/10/2016	12:23:10	2	1	TPM-3	9063
1315	-1	-1	Finished in success	FMA		0	End type	26/10/2016	12:23:07	26/10/2016	12:23:08	26/10/2016	12:23:08	0	1	fmarianne-pc	5022
1315	1	2	Finished in success	FMA		0	Main type	26/10/2016	12:23:07	26/10/2016	12:23:07	26/10/2016	12:23:07	0	1	fmarianne-pc	5023
1327	1	1	Finished in success	FMA		1221376818	Main type	26/10/2016	15:54:26	26/10/2016	15:54:26	26/10/2016	15:54:26	0	0	fmarianne-pc	5020
1327	-1	-1	Finished in success	FMA		1221376818	End type	26/10/2016	15:54:26	26/10/2016	15:54:27	26/10/2016	15:54:27	0	0	fmarianne-pc	5020
1326	1	1	Finished in success	FMA		0	Main type	26/10/2016	15:53:57	26/10/2016	15:53:57	26/10/2016	15:54:02	5	0	fmarianne-pc	5020
1326	-1	-1	Finished in success	FMA		0	End type	26/10/2016	15:53:57	26/10/2016	15:54:02	26/10/2016	15:54:02	0	0	fmarianne-pc	5020
1325	1	1	Ended in error	FMA	genord_2	0	Main type	26/10/2016	15:43:32	26/10/2016	16:41:35	26/10/2016	16:41:38	3	0	TPM-3	9062
1325	-1	-1	Inactive	FMA	genord_1	0	End type	26/10/2016	15:43:32					0	0	TPM-3	9061
1324	1	1	Finished in success	FMA		1522494105	Main type	26/10/2016	15:34:33	26/10/2016	15:34:33	26/10/2016	15:34:33	0	0	fmarianne-pc	5024

Figura 28: Captura del documento PDF generado al exportar el resultado de la búsqueda de una subtarea

5.4 Tareas

Se procede igualmente a mostrar la funcionalidad para el caso de una búsqueda de las tareas utilizando la función Display. La primera captura corresponde a la ventana de parámetros en la que el usuario introduce el valor de las variables con las que desea realizar la búsqueda. En este caso puede buscar las tareas en función prácticamente de la misma forma que una subtarea, el usuario que llevó a cabo la tarea y la fecha de creación de la tarea al igual que un intervalo de tiempo dentro del cual la tarea empezó. La segunda captura muestra lo que ve el usuario cuando intenta introducir un usuario, no se trata de una entrada libre sino que tiene que ser un usuario que exista en la base de datos, al introducir el usuario la plataforma nos muestra, en el caso de que haya encontrado uno, sus características. La tercera captura muestra el resultado, la tabla que se obtiene al realizar la búsqueda más básica, es decir una fecha cualquiera y todos los estados. En la cuarta imagen se aprecia una vista en detalle de una de las tareas, en ella se puede apreciar las subtareas creadas por el servidor de la misma forma que a través de su categoría, con la misma información y las mismas columnas lo que resulta muy práctico para entender como se ha dividido la tarea y a la hora de buscar subtareas. Finalmente se muestra una captura del documento en PDF que se obtiene cuando se exporta el resultado a dicho formato, la primera corresponde al PDF de las tareas y la segunda con el de la vista en detalle.

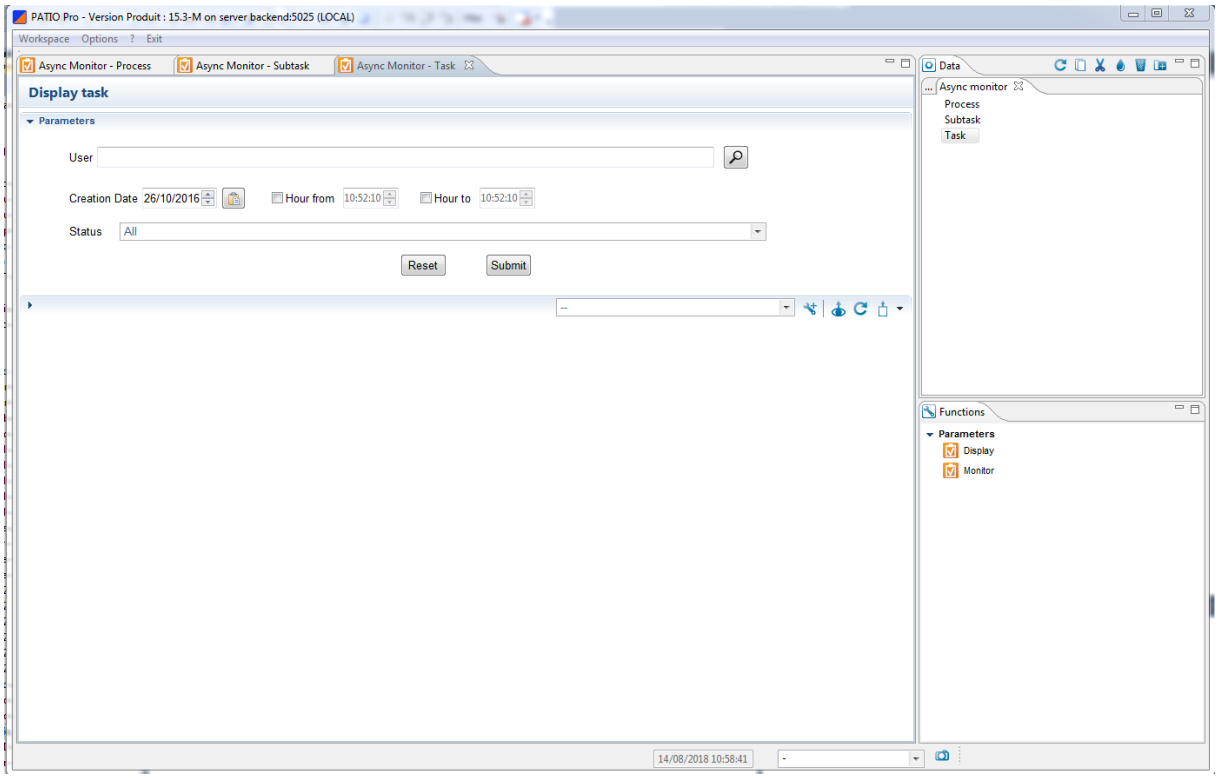


Figura 29: Ventana de parámetros de la función Display para una tarea

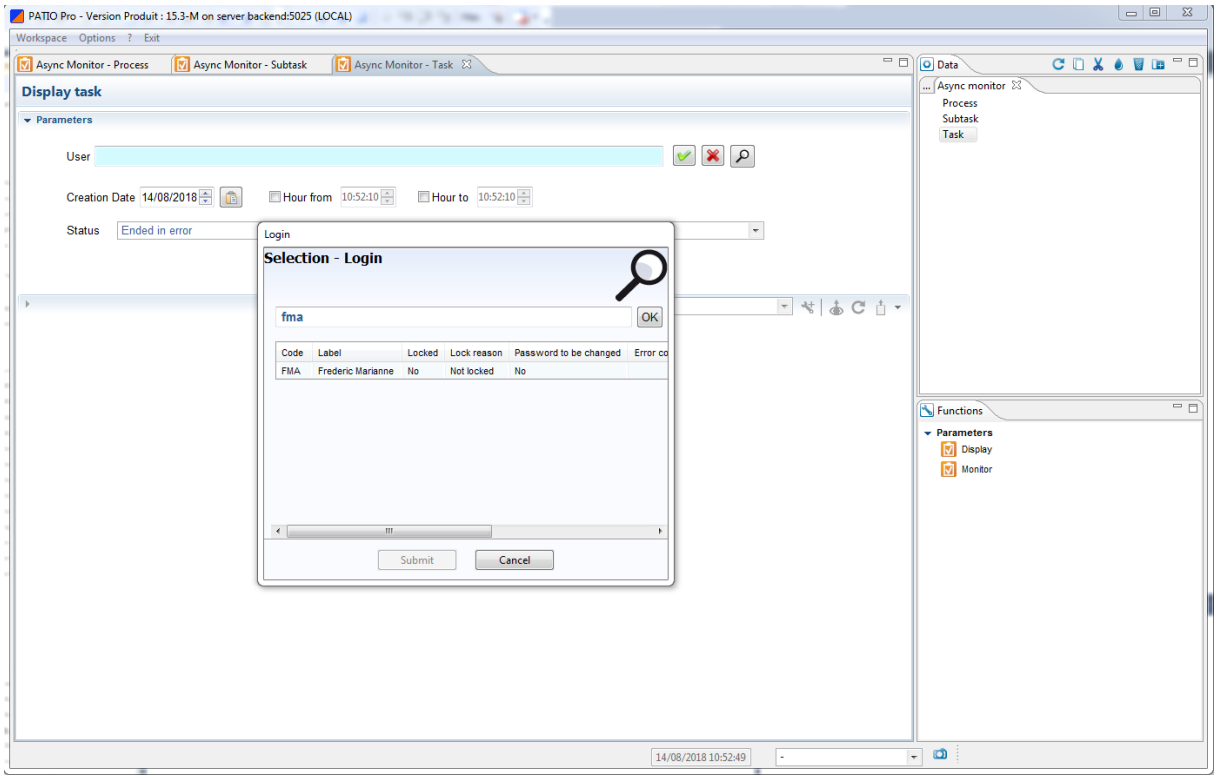


Figura 30: Comprobación del usuario introducido

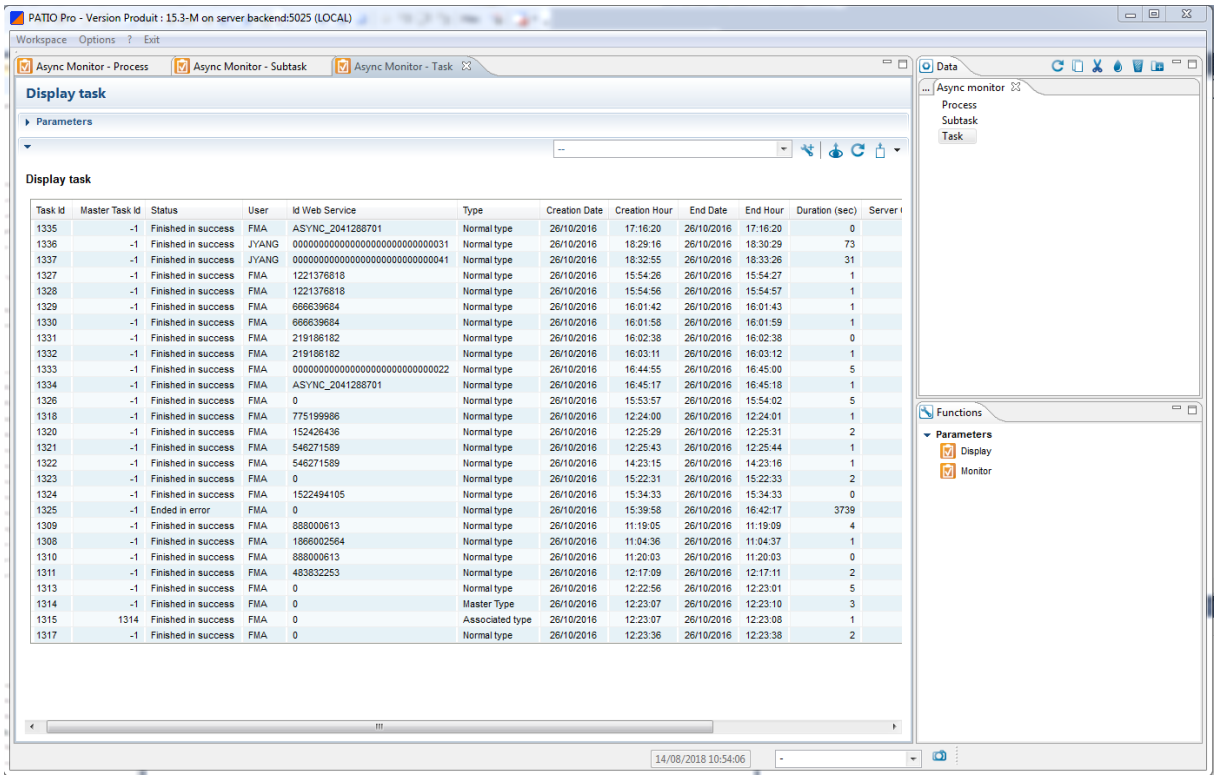


Figura 31: Resultado de la búsqueda de una tarea

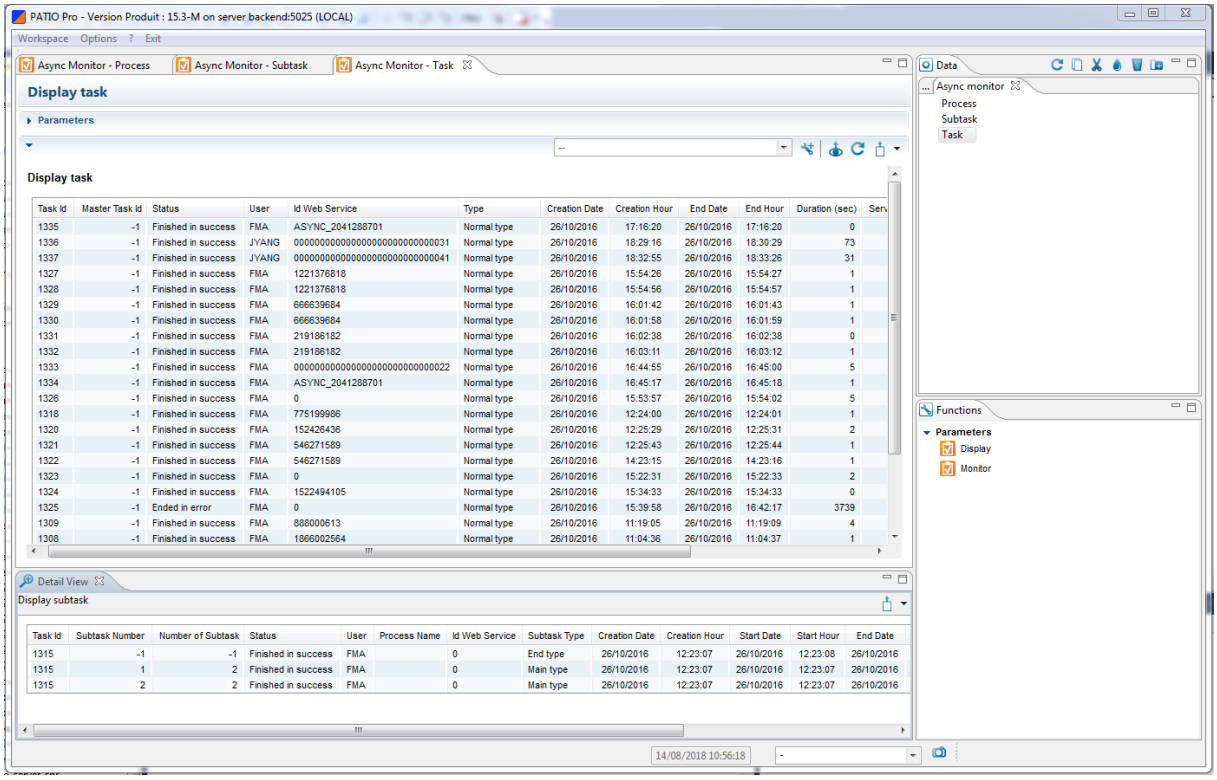


Figura 32: Resultado con vista en detalle de la búsqueda de una tarea

Task Id	Master Task Id	Status	User	Id Web Service	Type	Creation Date	Creation Hour	End Date	End Hour	Duration (sec)	Server Object	Request type	Notified	Subtask number
1335	-1	Finished in success	FMA	ASYNC_2041288701	Normal type	26/10/2016	17:16:20	26/10/2016	17:16:20	0	650	4	true	1
1336	-1	Finished in success	JYANG	00000000000000000000000000000031	Normal type	26/10/2016	18:29:16	26/10/2016	18:30:29	73	650	0	true	1
1337	-1	Finished in success	JYANG	00000000000000000000000000000041	Normal type	26/10/2016	18:32:55	26/10/2016	18:33:26	31	650	0	true	1
1327	-1	Finished in success	FMA	1221376818	Normal type	26/10/2016	15:54:26	26/10/2016	15:54:27	1	650	4	true	1
1328	-1	Finished in success	FMA	1221376818	Normal type	26/10/2016	15:54:56	26/10/2016	15:54:57	1	650	4	true	1
1329	-1	Finished in success	FMA	666639684	Normal type	26/10/2016	16:01:42	26/10/2016	16:01:43	1	650	4	true	1
1330	-1	Finished in success	FMA	666639684	Normal type	26/10/2016	16:01:58	26/10/2016	16:01:59	1	650	4	true	1
1331	-1	Finished in success	FMA	219186182	Normal type	26/10/2016	16:02:38	26/10/2016	16:02:38	0	650	4	true	1
1332	-1	Finished in success	FMA	219186182	Normal type	26/10/2016	16:03:11	26/10/2016	16:03:12	1	650	4	true	1
1333	-1	Finished in success	FMA	00000000000000000000000000000022	Normal type	26/10/2016	16:44:55	26/10/2016	16:45:00	5	650	0	true	1
1334	-1	Finished in success	FMA	ASYNC_2041288701	Normal type	26/10/2016	16:45:17	26/10/2016	16:45:18	1	650	4	true	1
1326	-1	Finished in success	FMA	0	Normal type	26/10/2016	15:53:57	26/10/2016	15:54:02	5	650	0	true	1
1318	-1	Finished in success	FMA	775199986	Normal type	26/10/2016	12:24:00	26/10/2016	12:24:01	1	650	4	true	1
1320	-1	Finished in success	FMA	152426436	Normal type	26/10/2016	12:25:29	26/10/2016	12:25:31	2	650	0	true	1
1321	-1	Finished in success	FMA	546271589	Normal type	26/10/2016	12:25:43	26/10/2016	12:25:44	1	650	4	true	1
1322	-1	Finished in success	FMA	546271589	Normal type	26/10/2016	14:23:15	26/10/2016	14:23:16	1	650	4	true	1
1323	-1	Finished in success	FMA	0	Normal type	26/10/2016	15:22:31	26/10/2016	15:22:33	2	650	0	true	1
1324	-1	Finished in success	FMA	1522494105	Normal type	26/10/2016	15:34:33	26/10/2016	15:34:33	0	650	4	true	1
1325	-1	Ended in error	FMA	0	Normal type	26/10/2016	15:39:58	26/10/2016	16:42:17	3739	650	0	true	1
1309	-1	Finished in success	FMA	888000613	Normal type	26/10/2016	11:19:05	26/10/2016	11:19:09	4	650	4	true	1
1308	-1	Finished in success	FMA	1866002564	Normal type	26/10/2016	11:04:36	26/10/2016	11:04:37	1	650	4	true	1
1310	-1	Finished in success	FMA	888000613	Normal type	26/10/2016	11:20:03	26/10/2016	11:20:03	0	650	4	true	1
1311	-1	Finished in success	FMA	483832253	Normal type	26/10/2016	12:17:09	26/10/2016	12:17:11	2	650	4	true	1
1313	-1	Finished in success	FMA	0	Normal type	26/10/2016	12:22:56	26/10/2016	12:23:01	5	650	0	true	1
1314	-1	Finished in success	FMA	0	Master Type	26/10/2016	12:23:07	26/10/2016	12:23:10	3	650	1	true	-1
1315	1314	Finished in success	FMA	0	Associated type	26/10/2016	12:23:07	26/10/2016	12:23:08	1	650	1	false	2
1317	-1	Finished in success	FMA	0	Normal type	26/10/2016	12:23:36	26/10/2016	12:23:38	2	650	0	true	1

Figura 33: Captura del documento PDF generado al exportar el resultado de la búsqueda de una tarea

Task Id	Subtask Number	Number of Subtask	Status	User	Process Name	Id Web Service	Subtask Type	Creation Date	Creation Hour	Start Date	Start Hour	End Date	End Hour	Duration (sec)	Complexit	Hostname	Port
1315	-1	-1	Finished in success	FMA		0	End type	26/10/201(	12:23:07	26/10/201(	12:23:08	26/10/201(	12:23:08	0	1	fmarianne-pc	5022
1315	1	2	Finished in success	FMA		0	Main type	26/10/201(	12:23:07	26/10/201(	12:23:07	26/10/201(	12:23:07	0	1	fmarianne-pc	5023
1315	2	2	Finished in success	FMA		0	Main type	26/10/201(	12:23:07	26/10/201(	12:23:07	26/10/201(	12:23:08	1	1	fmarianne-pc	5023

Figura 34: Captura del documento PDF generado al exportar la vista en detalle de una tarea

5.5 Monitor

Finalmente se muestra la función monitor, en este caso el usuario no realiza ninguna búsqueda simplemente lanza la consulta apretando el botón de la funcionalidad, la primera captura muestra el resultado obtenido y la segunda el documento PDF generado si se quiere exportar. Se aprecia tal y como se pidió en las especificaciones que cuando el número de servidores en error es superior a 0 el resultado sale de color rojo.

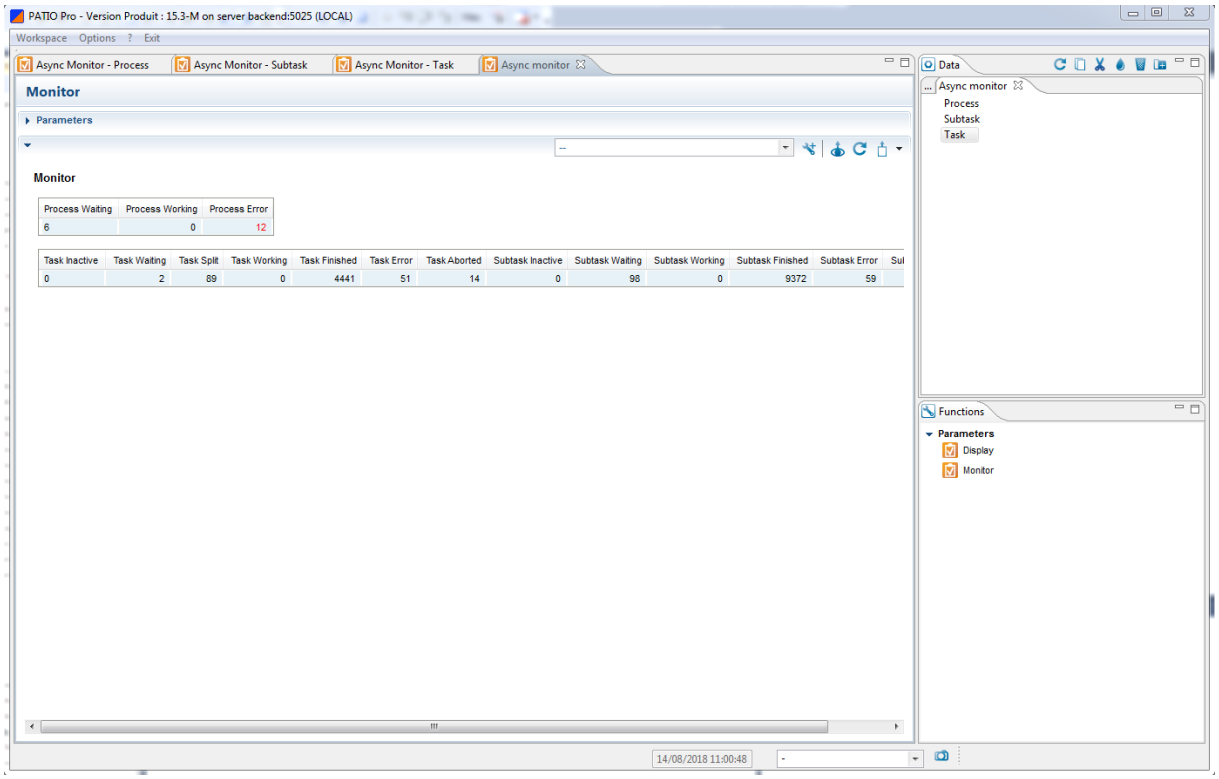


Figura 35: Resultado de la consulta del estado de los servidores, tareas y subtareas asíncronas

Process Waiting	Process Working	Process Error
6	0	12

Task Inactive	Task Waiting	Task Split	Task Working	Task Finished	Task Error	Task Aborted	Subtask Inactive	Subtask Waiting	Subtask Working	Subtask Finished	Subtask Error	Subtask Aborted
0	2	89	0	4441	51	14	0	98	0	9372	59	17

Figura 36: Captura del documento PDF generado al exportar el resultado de la consulta del estado de los servidores, tareas y subtareas asíncronas

6. Conclusión

6.1 Conclusión general

Con la aparición de las nuevas tecnologías nos hemos acostumbrado a recibir la información de forma instantánea, vivimos en la era de la inmediatez. Esto se aplica a todos los aspectos de la vida, tanto en el mundo laboral como en el día a día.

Ser capaces de ofrecer un servicio de forma rápida y eficaz se ha convertido por lo tanto en un desafío fundamental que todas las empresas deben afrontar.

Es aún más cierto en el mercado financiero y en particular en un servicio de ejecución de órdenes en tiempo real.

Esta es la idea principal en la que se basa este proyecto, que busca dotar a la empresa de una herramienta que le permita ser más rápida, eficaz y reactiva.

Una vez se ha concluido el proyecto se puede afirmar que el objetivo se ha cumplido. Gracias a las dos funciones que se han realizado se puede proporcionar al cliente una respuesta mucho más rápida sobre porque ha fallado su operación y a raíz de esto ofrecerle una solución en un plazo más corto. También permite llevar un seguimiento más preciso del estado de los servidores para poder corregir los fallos con mayor brevedad incluso antes de que el cliente tenga que contactar con la empresa, de forma que cuando lo haga ya se le pueda proporcionar una solución de forma inmediata.

Este aumento de la reactividad y disminución del tiempo de espera permite que el cliente reciba un mejor servicio, lo más rápido y eficaz posible.

6.2 Futuros desarrollos

La totalidad del proyecto se basa en realizar consultas a una base de datos para obtener visualmente la información que deseamos, todo esto se hace por lo tanto únicamente en lectura, el primer y más importante desarrollo que se le puede imaginar a este proyecto es por lo tanto extender esta característica. Podríamos imaginar por lo tanto no solo tener acceso a esta información sino también modificarla, esto nos permitiría actualizar ciertas informaciones de las tareas o de los servidores si vemos que no concuerdan con la realidad o directamente eliminar tareas de la base de datos si hemos tenido que cancelarlas por el motivo que sea. Sobre esta cuestión por el momento no se ha planteado nada ni se ha empezado a buscar posibles soluciones. Sin embargo ya existen ciertas funcionalidades que crean, actualizan, eliminan y permiten la visualización de información, con lo cual sería viable imaginar alguna solución parecida. Para ello, utilizando el código creado en este proyecto, se podrían añadir funciones en la parte servidor para actualizar o eliminar (en de cada uno de los tres códigos: servidor, tarea o subtarea) de forma equivalente a server_read. En JAVA sin duda habría que crear nuevas clases para ello.

Una posible mejora o continuación que ayudaría al equipo de soporte podría ser algún tipo de alarma o aviso que notificará cuando un servidor lleva más de x tiempo sin dar señales de vida, aunque su estado sea correcto. Esto va en el sentido de lo que ya se ha analizado previamente sobre el desfase existente entre el estado y el último heartbeat y va orientado a tener un mayor control en tiempo real de la actividad de los servidores, aunque se podría ampliar también a las tareas y subtareas avisando cuando una de ellas esté en error.

Siempre será posible añadir nuevos campos con los que realizar las búsquedas como por ejemplo intervalos en las horas de comienzo y de fin que faciliten la búsqueda cuando se tiene poca información. Al igual que se podrán imaginar nuevas funcionalidades ya que este proyecto es únicamente la primera versión y no ha sido todavía utilizada regularmente por el equipo de soporte realizando búsquedas para solucionar problemas reales. Solamente esta práctica podrá determinar si es necesario añadir cosas o si por el contrario ciertas partes no son realmente necesarias. Este proyecto forma por lo tanto la base y ahora queda perfeccionarla para que sea lo más precisa y práctica posible.

7. Referencias

[1] 4TPM. Documento Interno. “Patio Plaquette”

[2] 4TPM. Documento Interno. “Patio Plaquette”

[3] 4TPM. Documento Interno. “Patio Plaquette”

[4] Web de 4TPM.
Enlace. (http://4tpm.fr)

[5] Web de 4TPM
Enlace. (http://4tpm.fr/technologie-services/technologie/)

[6] 4TPM. Documento Interno. “Patio Plaquette”

[7] 4TPM. Documento Interno. “Patio Plaquette”

[8] Equipo de soporte 4TPM. Documento interno dedicado al equipo de soporte. “File d’attente support”

[9] Equipo de soporte 4TPM. Documento interno dedicado al equipo de soporte. “File d’attente support”

[10] Equipo de soporte 4TPM. Documento interno dedicado al equipo de soporte. “File d’attente support”

[11] Equipo de soporte 4TPM. Documento interno dedicado al equipo de soporte. “File d’attente support”

[12] Equipo de soporte 4TPM. Documento interno dedicado al equipo de soporte. “File d’attente support”

8. Anexos

A continuación se han incluido algunos ejemplos de programas realizados para este proyecto, tanto para la parte del servidor como para la parte cliente. Se ha optado por no incluir la totalidad de los programas realizados debido a que era demasiado extensa, más de 300 páginas de código. Por otro lado, en particular en lo relativo a la función Display, muchos de los programas se han tenido que realizar tres veces de forma muy similar, para los servidores, las tareas y finalmente las subtareas. Por esta razón en este anexo aparece simplemente un ejemplo para cada tipo de función.

Códigos del servidor:

En primer lugar se muestra el código de la función Monitor.

Monitor.c

```
+/*
+** ASYNC_MONITOR.C
+**
+** $Header$
+**
+**  Modifications History:
+** 13-AUG-2018 JN Add async task monitor screens in patio pro (standard object) (EVINT-3366).
+*/
+
+
+#define ASYNC_MONITOR_DEF
+
+#ifdef SERVER
+
+#include "sysenv.h"
+#include "sizedefs.h"
+#include "typedefs.h"
+
+#include "object.h"
+#include "seraudit.h"
+
+#include "async_monitor.h"
+#include "async_def.h"
+#include "datamess.h"
+
+static void async_monitor_server_read(ASYNC_MONITOR *, SOCKET);
+
+void dbasync_monitor_init()
+{
+       M_INIT_DESC2(ASYNC_MONITOR, ASYNC_MONITOR_ID, "PATIO_PATIO", "ASYNC_MONITOR", MESS_TAG_ASYNC_MONITOR)
+
+    M_DATA_EXTR("TYPEREQUESTI",                 SQL_SMALLINT,   SHORT,          typerequesti)
+    M_DATA_EXTR("TAGI",                         SQL_SMALLINT,   SHORT,          tagi)
+
+    M_DATA_EXTR("PROCESSWAITING",               SQL_INTEGER,  LONG,         proces swaiting)
+    M_DATA_EXTR("PROCESSWORKING",               SQL_INTEGER,  LONG,         proces sworking)
+    M_DATA_EXTR("PROCESSERROR",                 SQL_INTEGER,  LONG,         proces serror)
+    M_DATA_EXTR("TASKINACTIVE",                 SQL_INTEGER,  LONG,         taskin active)
+    M_DATA_EXTR("TASKWAITING",                  SQL_INTEGER,  LONG,         taskwa iting)
+    M_DATA_EXTR("TASKSPLIT",                    SQL_INTEGER,  LONG,         tasksp lit)
+    M_DATA_EXTR("TASKWORKING",                  SQL_INTEGER,  LONG,         taskwo rking)
+    M_DATA_EXTR("TASKFINISHED",                 SQL_INTEGER,  LONG,         taskfi nished)
+    M_DATA_EXTR("TASKERROR",                    SQL_INTEGER,  LONG,         tasker ror)
+    M_DATA_EXTR("TASKABORTED",                  SQL_INTEGER,  LONG,         taskab orted)
+    M_DATA_EXTR("SUBTASKINACTIVE",              SQL_INTEGER,  LONG,         subtas kinactive)
+    M_DATA_EXTR("SUBTASKWAITING",               SQL_INTEGER,  LONG,         subtas kwaiting)
```

```
47. +    M_DATA_EXTR("SUBTASKSPLIT",                SQL_INTEGER,  LONG,          subtas
    ksplit)
48. +    M_DATA_EXTR("SUBTASKWORKING",              SQL_INTEGER,  LONG,          subtas
    kworking)
49. +    M_DATA_EXTR("SUBTASKFINISHED",             SQL_INTEGER,  LONG,          subtas
    kfinished)
50. +    M_DATA_EXTR("SUBTASKERROR",                SQL_INTEGER,  LONG,          subtas
    kerror)
51. +    M_DATA_EXTR("SUBTASKABORTED",              SQL_INTEGER,  LONG,          subtas
    kaborted)
52. +
53. +    M_OBJECTREQUEST3_FP(async_monitor_from_objectrequest);
54. +
55. +    M_END_DESC()
56. +    M_END_DESC_CLI()
57. +    M_TABLE_NOT_IN_DATABASE()
58. +}
59. +
60. +/*
61. +** ASYNC_MONITOR_FROM_OBJECTREQUEST
62. +*/
63. +
64. +void async_monitor_from_objectrequest(void *datap, SOCKET socket)
65. +{
66. +    BLOCK block;
67. +    OBJECTREQUEST *objectrequestp = (OBJECTREQUEST *) datap;
68. +
69. +    if (objectrequestp -> typerequesti == TYPEREQUEST_READ)
70. +    {
71. +        block.blockheader.blocktagi = MESS_TAG_ASYNC_MONITOR;
72. +        block.datap = mem_get(database_desc[ASYNC_MONITOR_ID]->datasize);
73. +        objectrequestp->id_tablei = ASYNC_MONITOR_ID;
74. +    }
75. +
76. +    objectrequest_fill_request(objectrequestp, block.datap);
77. +    async_monitor_server_process(&block, socket);
78. +    mem_free(block.datap);
79. +}
80. +
81. +/*
82. +** ASYNC_MONITOR_SERVER_PROCESS
83. +*/
84. +
85. +void async_monitor_server_process(BLOCKPTR blockp, SOCKET socket)
86. +{
87. +    ASYNC_MONITOR *async_monitorp;
88. +    SHORT typerequesti;
89. +
90. +    if (blockp->blockheader.blocktagi == MESS_TAG_ASYNC_MONITOR)
91. +    {
92. +        async_monitorp = (ASYNC_MONITOR *) blockp->datap;
93. +        typerequesti = async_monitorp->typerequesti;
94. +    }
95. +
96. +    switch (typerequesti)
97. +    {
98. +        case TYPEREQUEST_READ :
99. +            async_monitor_server_read(async_monitorp, socket);
100.       +        break;
101.       +        default :
102.       +             seraudit_var("ERROR typerequesti not handled");
103.       +             /*  */
104.       +        break;
105.       +    }
106.       +}
107.       +
```

```
108.       +/*
109.       +** ASYNC_MONITOR_SERVER_READ
110.       +*/
111.       +
112.       +static void async_monitor_server_read(ASYNC_MONITOR *async_monitorp, SOCKET
    socket)
113.       +{
114.       +    static char* fctnames = "ASYNC_MONITOR_SERVER_READ";
115.       +    char wheres[1000];
116.       +    NODEPTR inventmonitor;
117.       +    ASYNC_MONITOR async_monitor;
118.       +    INT statusi = 0;
119.       +    long nb_per_status = 0L;
120.       +
121.       +    seraudit_varniv(NIVTRACE_DATABASE,"%s; BEGIN ", fctnames);
122.       +
123.       +    memset(wheres, '\0', sizeof(wheres));
124.       +
125.       +#if 0
126.       +/* TEST bind */
127.       +{
128.       +ASYNC_SUBTASK st;
129.       +memset(&st,'\0',sizeof(st));
130.       +strcpy(wheres,"where statusi < 0 ");
131.       +DBEXTREAD(ASYNC_SUBTASK_ID, 0, "select * ", 0, wheres, &st, 0, NULL, 0, NULL
    , 0, NULL, 0, NULL, 0, NULL, 0, NULL, 0, NULL, 0, 0, 0, 0, 0, 0)
132.       + {
133.       + seraudit_var("Debug statusi : %d",st.statusi);
134.       + }
135.       +
136.       +strcpy (wheres, " WHERE STATUSI < 0 GROUP BY STATUSI");
137.       +DBEXTREAD(ASYNC_SUBTASK_ID, 0, "select STATUSI, COUNT(1) as nb_per_status",
    0, wheres, NULL, 1, &statusi, 2, &nb_per_status, 0, NULL, 0, NULL, 0, NULL, 0, NULL,
     0, NULL, 0, 0, 0, 0, 0, 0)
138.       +{
139.       + seraudit_var("Debug statusi : %d",statusi);
140.       + }
141.       +}
142.       +/* END test bind */
143.       +#endif
144.       +
145.       +    /* tempo debug overflow INT -
    1 => 65535 so filtering out only positive values.*/
146.       +   strcpy (wheres, " WHERE STATUSI >= 0 GROUP BY STATUSI");
147.       +
148.       +    inventmonitor = node_list_init();
149.       +
150.       +    memset(&async_monitor, '\0', sizeof(async_monitor));
151.       +
152.       +    DBEXTREAD(ASYNC_TASK_ID, 0, "select STATUSI, COUNT(1) as nb_per_status",
     0, wheres, NULL, 1, &statusi, 2, &nb_per_status, 0, NULL, 0, NULL, 0, NULL, 0, NULL
    , 0, NULL, 0, 0, 0, 0, 0, 0)
153.       +    {
154.       +        switch (statusi)
155.       +        {
156.       +        case ASYNC_DEF_STATUS_INACTIVE:
157.       +        async_monitor.taskinactive = nb_per_status;
158.       +        break;
159.       +
160.       +        case ASYNC_DEF_STATUS_WAITING:
161.       +        async_monitor.taskwaiting = nb_per_status;
162.       +        break;
163.       +
164.       +        case ASYNC_DEF_STATUS_SPLIT:
165.       +        async_monitor.tasksplit = nb_per_status;
166.       +        break;
```

```
167.       +
168.       +         case ASYNC_DEF_STATUS_WORKING:
169.       +         async_monitor.taskworking = nb_per_status;
170.       +         break;
171.       +
172.       +         case ASYNC_DEF_STATUS_FINISHED:
173.       +         async_monitor.taskfinished = nb_per_status;
174.       +         break;
175.       +
176.       +         case ASYNC_DEF_STATUS_ERROR:
177.       +         async_monitor.taskerror = nb_per_status;
178.       +         break;
179.       +
180.       +         case ASYNC_DEF_STATUS_ABORTED:
181.       +         async_monitor.taskaborted = nb_per_status;
182.       +         break;
183.       +
184.       +         default : seraudit_var("Unknown status %d nb %ld ", statusi, nb_per_
    status);
185.       +         }
186.       +     }
187.       +
188.       +     DBEXTREAD(ASYNC_SUBTASK_ID, 0, "select STATUSI, COUNT(1) as nb_per_statu
    s", 0, wheres, NULL, 1, &statusi, 2, &nb_per_status, 0, NULL, 0, NULL, 0, NULL, 0, N
    ULL, 0, NULL, 0, 0, 0, 0, 0, 0)
189.       +     {
190.       +         switch (statusi)
191.       +         {
192.       +         case ASYNC_DEF_STATUS_INACTIVE:
193.       +         async_monitor.subtaskinactive = nb_per_status;
194.       +         break;
195.       +
196.       +         case ASYNC_DEF_STATUS_WAITING:
197.       +         async_monitor.subtaskwaiting = nb_per_status;
198.       +         break;
199.       +
200.       +         case ASYNC_DEF_STATUS_SPLIT:
201.       +         async_monitor.subtasksplit = nb_per_status;
202.       +         break;
203.       +
204.       +         case ASYNC_DEF_STATUS_WORKING:
205.       +         async_monitor.subtaskworking = nb_per_status;
206.       +         break;
207.       +
208.       +         case ASYNC_DEF_STATUS_FINISHED:
209.       +         async_monitor.subtaskfinished = nb_per_status;
210.       +         break;
211.       +
212.       +         case ASYNC_DEF_STATUS_ERROR:
213.       +         async_monitor.subtaskerror = nb_per_status;
214.       +         break;
215.       +
216.       +         case ASYNC_DEF_STATUS_ABORTED:
217.       +         async_monitor.subtaskaborted = nb_per_status;
218.       +         break;
219.       +
220.       +         default : seraudit_var("Unknown status %d nb %ld ", statusi, nb_per_
    status);
221.       +         }
222.       +     }
223.       +
224.       +     DBEXTREAD(ASYNC_REGISTRATION_ID, 0, "select STATUSI, COUNT(1) as nb_per_
    status", 0, wheres, NULL, 1, &statusi, 2, &nb_per_status, 0, NULL, 0, NULL, 0, NULL,
     0, NULL, 0, NULL, 0, 0, 0, 0, 0, 0)
225.       +     {
226.       +         switch (statusi)
```

```
227.       +        {
228.       +        case ASYNC_DEF_STATE_WAITING:
229.       +        async_monitor.processwaiting = nb_per_status;
230.       +        break;
231.       +
232.       +        case ASYNC_DEF_STATE_WORKING:
233.       +        async_monitor.processworking = nb_per_status;
234.       +        break;
235.       +
236.       +        case ASYNC_DEF_STATE_ZOMBIE:
237.       +        async_monitor.processerror = nb_per_status;
238.       +        break;
239.       +
240.       +        default : seraudit_var("Unknown status %d nb %ld ", statusi, nb_per_
    status);
241.       +        }
242.       +    }
243.       +
244.       +    node_list_add_elt(inventmonitor, &async_monitor, sizeof(async_monitor));

245.       +    memset(&async_monitor, '\0', sizeof(async_monitor));
246.       +
247.       +    object_server_invent_send_bundle(ASYNC_MONITOR_ID, NULL, inventmonitor,
    socket);
248.       +
249.       +    node_list_dispose(inventmonitor);
250.       +
251.       +}
252.       +
253.       +#endif /* End #ifdef SERVER*/
254.       +/* EOF */
```

Se sigue mostrando el archivo de cabecera de la función Monitor.

Monitor.h

```
1.  +/*
2.  +** ASYNC_MONITOR.H
3.  +**
4.  +** $Header$
5.  +**
6.  +**  Modifications History:
7.  +** 05-JUL-2018 JNA Creation.
8.  +*/
9.  +
10. +#ifdef ASYNC_MONITOR_DEF
11. +#define ASYNC_MONITOR_EXTDEF
12. +#else
13. +#define ASYNC_MONITOR_EXTDEF extern
14. +#endif
15. +
16. +typedef struct __async_monitor__
17. +{
18. +    short typerequesti;
19. +    short tagi;
20. +
21. +    //Processus
22. +    long processwaiting;
23. +    long processworking;
24. +    long processerror;
25. +
26. +    //Task
27. +    long taskinactive;
```

```
28. +    long taskwaiting;
29. +    long tasksplit;
30. +    long taskworking;
31. +    long taskfinished;
32. +    long taskerror;
33. +    long taskaborted;
34. +
35. +    //Subtask
36. +    long subtaskinactive;
37. +    long subtaskwaiting;
38. +    long subtasksplit;
39. +    long subtaskworking;
40. +    long subtaskfinished;
41. +    long subtaskerror;
42. +    long subtaskaborted;
43. +
44. +} ASYNC_MONITOR;
45. +
46. +#ifdef SERVER
47. +
48. +ASYNC_MONITOR_EXTDEF void dbasync_monitor_init();
49. +
50. +ASYNC_MONITOR_EXTDEF void async_monitor_server_process(BLOCKPTR, SOCKET);
51. +ASYNC_MONITOR_EXTDEF void async_monitor_from_objectrequest(void *, SOCKET);;
52. +
53. +#endif /* End #ifdef SERVER*/
54. +
55. +/* EOF */
```

Código de la parte cliente:

En primer lugar se incluye el código de la clase AsyncMonitorDisplayProcessDataSet:

```
1.  /**
2.  + * AsyncMonitorDisplayProcessDataSet.java <br>
3.  + * <br>
4.  + * Description : TODO <TO BE FILLED>
5.  + * <br>
6.  + * <br>
7.  + * Modifications history :<br>
8.  + * <br>
9.  + * TODO <DD-MMM-YYYY> Creation <br>
10. + * <br>
11. + */
12. +public class AsyncMonitorDisplayProcessDataSet extends TPMDataSet<List<AsyncMonitor
    Process>, AsyncMonitorProcess>
13. +{
14. +    private AsyncMonitorProcessStatus asyncMonitorProcessStatus;
15. +    /**
16. +     *
17. +     */
18. +    public AsyncMonitorDisplayProcessDataSet()
19. +    {
20. +        super(I18nUtil.MSG_BUNDLE);
21. +    }
22. +
23. +    @Override
24. +    protected List<ReportColumnHolder> getColumns(String tableName)
25. +    {
26. +        reportColumnHolders
```

```
+                .add(new ReportColumnHolder("hostname", "async.monitor.process.hostname", TPMColumnType.STRING, true, true));
+        reportColumnHolders.add(new ReportColumnHolder("port", "async.monitor.process.port", TPMColumnType.INTEGER, true, true));
+        reportColumnHolders
+                .add(new ReportColumnHolder("processname", "async.monitor.process.processname", TPMColumnType.STRING, true, true));
+        reportColumnHolders.add(new ReportColumnHolder("lastheartbeatdate", "async.monitor.process.lastheartbeatdate",
+                TPMColumnType.DATE, true, true));
+        reportColumnHolders.add(new ReportColumnHolder("lastheartbeathour", "async.monitor.process.lastheartbeathour",
+                TPMColumnType.TIME, true, true));
+        reportColumnHolders.add(new ReportColumnHolder("status", "async.monitor.process.status",
+                TPMColumnType.STRING, true, true));
+        reportColumnHolders
+                .add(new ReportColumnHolder("complexity", "async.monitor.process.complexity", TPMColumnType.INTEGER, true, true));
+
+        return reportColumnHolders;
+    }
+
+    @Override
+    protected void performOpen(List<AsyncMonitorProcess> data)
+    {
+        result = data;
+    }
+
+    @Override
+    protected void performFetch(IUpdatableDataSetRow row, AsyncMonitorProcess item)
     throws ScriptException
+    {
+        setRowColumnValue(row, "hostname", item.getServer_names(), item.getClass().getSimpleName());
+        setRowColumnValue(row, "port", item.getServer_portl(), item.getClass().getSimpleName());
+        setRowColumnValue(row, "processname", item.getClient_names(), item.getClass().getSimpleName());
+        setRowColumnValue(row, "lastheartbeatdate", item.getHeartbeat_datel(), DateType.DATE, item.getClass().getSimpleName());
+        setRowColumnValue(row, "lastheartbeathour", item.getHeartbeat_datel(), DateType.TIME_HHMMSS,
+                item.getClass().getSimpleName());
+        setRowColumnValue(row, "status", I18nUtil.getI18nLabel(item.getStatus()), item.getClass().getSimpleName());
+        setRowColumnValue(row, "complexity", item.getComplexityi(), item.getClass().getSimpleName());
+
+    }
+
+}
```

A continuación se muestra el código de la clase DisplayAsyncMonitorTaskEditor:

```
+/**
+ * DisplayEditor.java <br>
+ * <br>
+ * Description : TODO <TO BE FILLED>
+ * <br>
+ * <br>
+ * Modifications history :<br>
+ * <br>
+ * TODO <DD-MMM-YYYY> Creation <br>
+ * <br>
```

```
+ */
+public class DisplayAsyncMonitorTaskEditor extends DisplayAsyncMonitorEditor
+{
+    private TPMCalendar                          startDate;
+    protected InputControl<Login>                loginControl;
+    protected Label                              loginControlLabel;
+    private DateTime                             timeWidgetStart;
+    private DateTime                             timeWidgetEnd;
+    private TPMEnumCombo<AsyncMonitorTaskStatus> combo;
+    private Button                               activateValidityStartTime;
+    private Button                               activateValidityEndTime;
+    private AsyncMonitorTaskRequest              query;
+    private List<AsyncMonitorTask>               asyncMonitorTasks;
+    private DisplayAsyncMonitorTaskResult        asyncMonitorTaskResult;
+    private static final int                     INDENT = 15;
+    private static final int                     WIDTH  = 200;
+    private GridData                             gridData;
+
+    @Override
+    protected void buildParametersInputZoneContent(Composite parent, FormToolkit to
olkit)
+    {
+        // User
+        parent.setLayout(new GridLayout());
+        Composite UserComposite = IhmUtilHolder.getIhmUtil().createComposite(parent
, SWT.NONE, "NoBackground");
+        gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
+        gridData.horizontalIndent = 40;
+        UserComposite.setLayoutData(gridData);
+        UserComposite.setLayout(new GridLayout(3, false));
+
+        loginControlLabel = IhmUtilHolder.getIhmUtil().createLabel(UserComposite, S
WT.NONE,
+                I18nUtil.getI18nMessage("display.asyncmonitor.user"), null, null);

+
+        gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
+        loginControlLabel.setLayoutData(gridData);
+
+        loginControl = new InputControl<Login>(new LoginIHMAdapter(), UserComposite
, true, false, null);
+        GridData grid = new GridData(SWT.FILL, SWT.CENTER, true, false);
+        grid.widthHint = 765;
+        loginControl.setLayoutData(grid);
+
+        // Date
+
+        AsyncMonitorTaskFilterByDate(parent, this);
+
+        // Combobox
+        parent.setLayout(new GridLayout());
+        Composite ComboComposite = IhmUtilHolder.getIhmUtil().createComposite(paren
t, SWT.NONE, "NoBackground");
+        gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
+        gridData.horizontalIndent = 40;
+        ComboComposite.setLayoutData(gridData);
+        ComboComposite.setLayout(new GridLayout(2, false));
+
+        Label status = IhmUtilHolder.getIhmUtil().createLabel(ComboComposite, SWT.N
ONE,
+                I18nUtil.getI18nMessage("asyncmonitor.task.combo.title"), null, nul
l);
+        gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
+        status.setLayoutData(gridData);
+
```

```
68. +        combo = (TPMEnumCombo<AsyncMonitorTaskStatus>) IhmUtilHolder.getIhmUtil().c
    reateTPMCombo(ComboComposite, SWT.NONE,
69. +                AsyncMonitorTaskStatus.values(), false, false, null, null);
70. +        gridData = new GridData(SWT.FILL, SWT.CENTER, true, true);
71. +        combo.setLayoutData(gridData);
72. +        combo.enableTooltips();
73. +        gridData = new GridData(SWT.FILL, SWT.CENTER, true, true);
74. +        gridData.widthHint = WIDTH;
75. +        gridData.horizontalIndent = INDENT;
76. +        combo.setLayoutData(gridData);
77. +        combo.setSelectedValue(AsyncMonitorTaskStatus.ERROR);
78. +
79. +    }
80. +
81. +    public boolean AsyncMonitorTaskFilterByDate(final Composite parent, final Scena
    rioEditorWithParams editor)
82. +    {
83. +        parent.setLayout(new GridLayout());
84. +        Composite periodComposite = IhmUtilHolder.getIhmUtil().createComposite(pare
    nt, SWT.NONE, "NoBackground");
85. +        GridData gridData = new GridData(SWT.FILL, SWT.CENTER, false, true);
86. +        gridData.horizontalIndent = 40;
87. +        periodComposite.setLayoutData(gridData);
88. +        periodComposite.setLayout(new GridLayout(9, false));
89. +
90. +        IhmUtilHolder.getIhmUtil().createLabel(periodComposite, SWT.NONE,
91. +                I18nUtil.getI18nMessage("displayasincmonitor.filterby.period.from")
    , null, null);
92. +
93. +        startDate = new TPMCalendar(periodComposite, SWT.NONE, TableWrapData.MIDDLE
    , editor, false);
94. +
95. +        activateValidityStartTime = IhmUtilHolder.getIhmUtil().createButton(periodC
    omposite, SWT.CHECK,
96. +                I18nUtil.getI18nMessage("displayasincmonitor.filterby.time.from"),
    "NoBackground", null, null);
97. +        gridData = new GridData(SWT.LEFT, SWT.CENTER, false, true);
98. +        gridData.horizontalIndent = 20;
99. +        activateValidityStartTime.setLayoutData(gridData);
100.        +
101.        +        timeWidgetStart = new DateTime(periodComposite, SWT.TIME | SWT.BORDE
    R);
102.        +        timeWidgetStart.setLayoutData(new GridData(SWT.LEFT, SWT.CENTER, fal
    se, false));
103.        +
104.        +        activateValidityStartTime.addSelectionListener(new SelectionAdapter(
    )
105.        +        {
106.        +            @Override
107.        +            public void widgetSelected(final SelectionEvent e)
108.        +            {
109.        +                Calendar calendar = Calendar.getInstance();
110.        +                timeWidgetStart.setTime(calendar.get(Calendar.HOUR_OF_DAY),
    calendar.get(Calendar.MINUTE),
111.        +                        calendar.get(Calendar.SECOND));
112.        +                timeWidgetStart.setEnabled(activateValidityStartTime.getSele
    ction());
113.        +            }
114.        +        });
115.        +        activateValidityStartTime.setSelection(false);
116.        +        activateValidityStartTime.notifyListeners(SWT.Selection, new Event()
    );
117.        +
118.        +        activateValidityEndTime = IhmUtilHolder.getIhmUtil().createButton(pe
    riodComposite, SWT.CHECK,
```

```
119.       +                I18nUtil.getI18nMessage("displayasincmonitor.filterby.time.to"), "NoBackground", null, null);
120.       +        gridData = new GridData(SWT.LEFT, SWT.CENTER, false, true);
121.       +        gridData.horizontalIndent = 20;
122.       +        activateValidityEndTime.setLayoutData(gridData);
123.       +
124.       +        timeWidgetEnd = new DateTime(periodComposite, SWT.TIME | SWT.BORDER);
125.       +        timeWidgetEnd.setLayoutData(new GridData(SWT.LEFT, SWT.CENTER, false, false));
126.       +
127.       +        activateValidityEndTime.addSelectionListener(new SelectionAdapter()
128.       +        {
129.       +            @Override
130.       +            public void widgetSelected(final SelectionEvent e)
131.       +            {
132.       +                Calendar calendar = Calendar.getInstance();
133.       +                timeWidgetEnd.setTime(calendar.get(Calendar.HOUR_OF_DAY), calendar.get(Calendar.MINUTE),
134.       +                        calendar.get(Calendar.SECOND));
135.       +                timeWidgetEnd.setEnabled(activateValidityEndTime.getSelection());
136.       +            }
137.       +        });
138.       +        activateValidityEndTime.setSelection(false);
139.       +        activateValidityEndTime.notifyListeners(SWT.Selection, new Event());
140.       +
141.       +        return true;
142.       +    }
143.       +
144.       +    @Override
145.       +    protected void performValidate()
146.       +    {
147.       +        query = new AsyncMonitorTaskRequest();
148.       +
149.       +        query.setStatus(combo.getSelectedValue());
150.       +
151.       +        TPMDate date = startDate.getTPMDate();
152.       +        if (activateValidityStartTime.getSelection())
153.       +        {
154.       +            Calendar calendar = Calendar.getInstance();
155.       +            calendar.set(date.getCalendar().get(Calendar.YEAR), date.getCalendar().get(Calendar.MONTH),
156.       +                    date.getCalendar().get(Calendar.DATE), timeWidgetStart.getHours(), timeWidgetStart.getMinutes(),
157.       +                    timeWidgetStart.getSeconds());
158.       +            date.setType(DateType.DATE_TIME);
159.       +            date.setDate(calendar.getTime());
160.       +        }
161.       +        query.setDtcreal(date);
162.       +
163.       +        TPMDate dateEnd = startDate.getTPMDate();
164.       +        if (activateValidityEndTime.getSelection())
165.       +        {
166.       +            Calendar calendar = Calendar.getInstance();
167.       +            calendar.set(dateEnd.getCalendar().get(Calendar.YEAR), dateEnd.getCalendar().get(Calendar.MONTH),
168.       +                    dateEnd.getCalendar().get(Calendar.DATE), timeWidgetEnd.getHours(), timeWidgetEnd.getMinutes(),
169.       +                    timeWidgetEnd.getSeconds());
170.       +            dateEnd.setType(DateType.DATE_TIME);
171.       +            dateEnd.setDate(calendar.getTime());
172.       +            query.setDtendl(dateEnd);
173.       +        }
```

```
+
+        if (!loginControl.getDataList().isEmpty())
+        {
+            Login login = (Login) loginControl.getDataList().get(0);
+            query.setLogin(login);
+        }
+
+
+    }
+
+    @Override
+    public void invokeServiceOnValidate(TPMUserInfo userInfo) throws TPMServ
iceException
+    {
+        asyncMonitorTasks = ServiceFactory.getBlockOrderService().asyncMonit
orTask(userInfo, query);
+    }
+
+    @Override
+    public String getEditorTitle()
+    {
+        return I18nUtil.getI18nMessage("display.asyncmonitortask.editor.titl
e");
+    }
+
+    @Override
+    public String getTitleToolTip()
+    {
+        return getEditorTitle();
+    }
+
+    @Override
+    protected void addScenarioResults()
+    {
+        asyncMonitorTaskResult = new DisplayAsyncMonitorTaskResult(this);
+        results.add(asyncMonitorTaskResult);
+
+    }
+
+    @Override
+    protected void setDataAndParameters()
+    {
+        asyncMonitorTaskResult.setData(asyncMonitorTasks);
+    }
+
+    @Override
+    protected boolean canRenderResult()
+    {
+        return asyncMonitorTasks != null;
+    }
+
+    @Override
+    public void inputChanged()
+    {
+        super.inputChanged();
+        this.setPartName(I18nUtil.getI18nMessage("display.asyncmonitortask.t
itle"));
+    }
+
+}
```

DisplayAsyncMonitorTaskResult:

```
1.  **
```

```
2.  + * DisplayAsyncMonitorTaskResult.java <br>
3.  + * <br>
4.  + * Description : TODO <TO BE FILLED>
5.  + * <br>
6.  + * <br>
7.  + * Modifications history :<br>
8.  + * <br>
9.  + * TODO <DD-MMM-YYYY> Creation <br>
10. + * <br>
11. + */
12. +public class DisplayAsyncMonitorTaskResult extends ScenarioSWTResult<List<AsyncMoni
    torTask>>
13. +        implements AddExtraDataDetailViewOpener, AddExtraDataDetailViewUpdater
14. +{
15. +    private static final Logger       LOGGER = LoggerFactory.getLogger(DisplayAsync
    MonitorTaskResult.class);
16. +    private List<AsyncMonitorSubtask> asyncMonitorSubtasks;
17. +
18. +    public DisplayAsyncMonitorTaskResult(final ScenarioPart scenarioPart)
19. +    {
20. +        super(scenarioPart);
21. +    }
22. +
23. +    @Override
24. +    public String getMnemoForExport()
25. +    {
26. +        return "AMT";
27. +    }
28. +
29. +    @Override
30. +    protected void buildResultContent(Composite parent, List<AsyncMonitorTask> data
    , FormToolkit toolkit, SWTViewer swtViewer)
31. +    {
32. +        TableData tableData = new TableData(3230, AsyncMonitorDisplayTaskDataSet.cl
    ass.getName(), "tableAsyncMonitorTask",
33. +                "asyncDisplaySubtask.rptdesign");
34. +        Composite composite = IhmUtilHolder.getIhmUtil().createComposite(parent, SW
    T.NONE, null);
35. +        composite.setLayout(new GridLayout());
36. +        swtViewer.buildTableContent(composite, tableData);
37. +
38. +    }
39. +
40. +    @Override
41. +    protected String getResultTitle(List<AsyncMonitorTask> data)
42. +    {
43. +        return this.getEditorTitle();
44. +    }
45. +
46. +    @Override
47. +    protected String getRptDesignName()
48. +    {
49. +        return "asyncDisplayTask.rptdesign";
50. +    }
51. +
52. +    @Override
53. +    protected Object computeReportData(List<AsyncMonitorTask> data)
54. +    {
55. +        return data;
56. +    }
57. +
58. +    @Override
59. +    protected Map<String, Object> computeReportParameters(List<AsyncMonitorTask> da
    ta)
60. +    {
61. +        return new HashMap<String, Object>();
```

```
62. +    }
63. +
64. +    @Override
65. +    public boolean isDataEmpty()
66. +    {
67. +        return data == null || data.isEmpty();
68. +    }
69. +
70. +    protected boolean isWithDetail()
71. +    {
72. +        return true;
73. +    }
74. +
75. +
76. +    @Override
77. +    public void addExtraData(ReportRowHolder row)
78. +    {
79. +        if (row.getData() != null)
80. +        {
81. +            IWorkbenchPage page = PlatformUI.getWorkbench().getActiveWorkbenchWindo
    w().getActivePage();
82. +            DetailView detailView = (DetailView) page.findView(DetailView.ID);
83. +            detailView.setContentDescription(I18nUtil.getI18nMessage("display.async
    monitorsubtask.editor.title"));
84. +
85. +            AsyncMonitorTask TaskInput = (AsyncMonitorTask) row.getData();
86. +            AsyncMonitorSubtaskRequest SubtaskRequest = new AsyncMonitorSubtaskRequ
    est();
87. +            SubtaskRequest.setid_taskl(TaskInput.getId_taskl());
88. +            SubtaskRequest.setStatus(AsyncMonitorTaskStatus.ALL);
89. +
90. +            try
91. +            {
92. +                asyncMonitorSubtasks = ServiceFactory.getBlockOrderService()
93. +                        .asyncMonitorSubtask(UIUserContext.getCurrent().getUserInfo
    (), SubtaskRequest);
94. +            }
95. +            catch (TPMServiceException e)
96. +            {
97. +                LOGGER.error("Failed to find one async monitor subtask", e);
98. +            }
99. +
100.       +            row.setDetailViewData((Object) asyncMonitorSubtasks);
101.       +
102.       +        }
103.       +    }
104.       +
105.       +}
```

OpenAsyncMonitorDiplayTask

```
1.  /**
2.  + * OpenAsyncMonitorDisplay.java <br>
3.  + * <br>
4.  + * Description : TODO <TO BE FILLED> <br>
5.  + * <br>
6.  + * Modifications history :<br>
7.  + * <br>
8.  + * TODO <DD-MMM-YYYY> Creation <br>
9.  + * <br>
10. + */
11. +public class OpenAsyncMonitorDisplayTask extends OpenAsyncMonitorDisplay
12. +{
13. +    @Override
14. +    public String getEditorID()
```

```
+    {
+        return EditorUtil.ASYNCMONITORTASK_EDITOR_ID;
+    }
+
+    @Override
+    protected String getEditorFunction()
+    {
+        return I18nUtil.getI18nMessage(EditorUtil.ASYNCMONITORTASK_FUNCTION_NAME);

+    }
+
+}
```

AsyncMonitorSubtaskAdapter:

```
/**
 * AsyncMonitorSubtaskAdapter.java <br>
 * <br>
 * Description : TODO <TO BE FILLED>
 * <br>
 * <br>
 * Modifications history :<br>
 * <br>
 * TODO <DD-MMM-YYYY> Creation <br>
 * <br>
 */
public class AsyncMonitorSubtaskAdapter extends TPMAdapter<AsyncMonitorSubtask>
{

    @Override
    public APIJAVAOBJSTD convertToPatioObj(TPMUserInfo userInfo, AsyncMonitorSubtask data)
    {
        return null;
    }

    @Override
    public APIJAVAOBJSTD convertToPatioObjRequest(TPMUserInfo userInfo, AsyncMonitorSubtask data)
    {
        return null;
    }

    @Override
    public AsyncMonitorSubtask convertFromPatioObj(APIJAVAOBJSTD patioObj, TPMUserInfo userInfo)
    {
        ASYNC_SUBTASK async_SUBTASK = (ASYNC_SUBTASK) patioObj;
        AsyncMonitorSubtask subtask = new AsyncMonitorSubtask();

        subtask.setId_taskl(async_SUBTASK.getId_taskl());
        subtask.setId_numberi(async_SUBTASK.getId_numberi());
        subtask.setSubtask_totali(async_SUBTASK.getSubtask_totali());
        subtask.setDtcreal(new TPMDate(async_SUBTASK.getDtcreal(), async_SUBTASK.getHrcreal(), DateType.DATE_TIME,
                CONFIG.getServerTimeZone()));
        subtask.setDtstartl(new TPMDate(async_SUBTASK.getDtstartl(), async_SUBTASK.getHrstartl(), DateType.DATE_TIME,
                CONFIG.getServerTimeZone()));
        subtask.setDtendl(
                new TPMDate(async_SUBTASK.getDtendl(), async_SUBTASK.getHrendl(), DateType.DATE_TIME, CONFIG.getServerTimeZone()));
        subtask.setDurationl(async_SUBTASK.getDurationl());
        subtask.setWeb_service_ids(async_SUBTASK.getWeb_service_ids());
        subtask.setComplexityi(async_SUBTASK.getComplexityi());
```

```
45. +        subtask.setStatus(AsyncMonitorTaskStatus.getEnum(async_SUBTASK.getStatusi()
    ));
46. +        subtask.setClient_names(async_SUBTASK.getClient_names());
47. +        subtask.setServer_names(async_SUBTASK.getServer_names());
48. +        subtask.setServer_portl(async_SUBTASK.getServer_portl());
49. +        subtask.setHeartbeat_datel(async_SUBTASK.getHeartbeat_datel());
50. +        subtask.setHeartbeat_hourl(async_SUBTASK.getHeartbeat_hourl());
51. +        subtask.setHeartbeat_timestampl(async_SUBTASK.getHeartbeat_timestampl());
52. +        subtask.setReservedb(async_SUBTASK.getReservedb() == 1);
53. +        subtask.setSubtask_typei(AsyncMonitorSubtaskType.getEnum(async_SUBTASK.getS
    ubtask_typei()));
54. +        subtask.setTask_logins(async_SUBTASK.getTask_logins());
55. +
56. +        return subtask;
57. +    }
58. +
59. +}
```

BlockOrderServiceImpl función subtask:

```
1.      public List<AsyncMonitorSubtask> asyncMonitorSubtask(final TPMUserInfo userInfo,
     final AsyncMonitorSubtaskRequest query)
2.  +            throws TPMServiceException
3.  +    {
4.  +
5.  +        return (List<AsyncMonitorSubtask>) arbTemplate.execute(userInfo, new ArbCal
    lBack()
6.  +        {
7.  +
8.  +            @Override
9.  +            public List<AsyncMonitorSubtask> doInPatio(ActioRequestBroker arb) thro
    ws TPMServiceException
10. +            {
11. +                arbTemplate.setStart(System.currentTimeMillis());
12. +
13. +                OBJECTREQUEST objectrequest = new OBJECTREQUEST(new ASYNC_SUBTASKRE
    QUEST());
14. +                objectrequest.setTypeRequesti(SYSENV.TYPEREQUEST_READ);
15. +                if (query != null)
16. +                {
17. +                    if (query.getStatus() != null)
18. +                    {
19. +                        objectrequest.setParam("statusi", query.getStatus().getValu
    e());
20. +                    }
21. +                    if (query.getDtcreal() != null)
22. +                    {
23. +                        TPMAdapter.makePatioDateForServer(query.getDtcreal(), objec
    trequest, "dtcreal", "hrcreal");
24. +                    }
25. +                    if (query.getDtendl() != null)
26. +                    {
27. +                        TPMAdapter.makePatioDateForServer(query.getDtendl(), object
    request, null, "hrendl");
28. +                    }
29. +                    if (query.getTask_logins() != null)
30. +                        objectrequest.setParam("task_logins", query.getTask_logins(
    ).getCode());
31. +                    if (query.getid_taskl() != 0)
32. +                        objectrequest.setParam("id_taskl", query.getid_taskl());
33. +                }
34. +
35. +                arbTemplate.setBeforeCallingPatio(System.currentTimeMillis());
36. +                ActioList result;
```

```
+                try
+                {
+                    result = arb.getActioList(objectrequest, (int) arb.getTimeout()
);
+                }
+                catch (PatioException e)
+                {
+                    TPMServiceException exc;
+                    exc = CommonServiceImpl.defineServiceException(e);
+                    throw exc;
+                }
+                arbTemplate.setAfterCallingPatio(System.currentTimeMillis());
+
+                // Conversion result api object in service object
+                AsyncMonitorSubtaskAdapter adapter = new AsyncMonitorSubtaskAdapter
();
+                List<AsyncMonitorSubtask> asyncMonitorSubtasks = new ArrayList<Asyn
cMonitorSubtask>();
+                Iterator it = result.getIterator();
+                while (it.hasNext())
+                    asyncMonitorSubtasks.add(adapter.convertFromPatioObj((APIJAVAOB
JSTD) it.next(), userInfo));
+
+                arbTemplate.setStop(System.currentTimeMillis());
+                SharedUtil.trace(LOGGER, arbTemplate.getStart(), arbTemplate.getBef
oreCallingPatio(),
+                        arbTemplate.getAfterCallingPatio(), arbTemplate.getStop(),
"asyncMonitorSubtask", "BlockOrderServiceImpl",
+                        arb.getServerName(), arb.getServerPort(), arb.getCurrentTra
nsactionId());
+
+                return asyncMonitorSubtasks;
+            }
+        });
+    }
```

Enume: AsyncMonitorTaskStatus, permite asociar el nombre del estado a su valor:

```
1.  /**
2.  + * TaskStatus.java <br>
3.  + * <br>
4.  + * Description : TODO <TO BE FILLED>
5.  + * <br>
6.  + * <br>
7.  + * Modifications history :<br>
8.  + * <br>
9.  + * TODO <DD-MMM-YYYY> Creation <br>
10. + * <br>
11. + */
12. +public enum AsyncMonitorTaskStatus
13. +{
14. +    /** Status inactive */
15. +    INACTIVE(-1),
16. +    /** Status waiting */
17. +    WAITING(0),
18. +    /** Status split */
19. +    SPLIT(1),
20. +    /** Status working */
21. +    WORKING(2),
22. +    /** Status finished */
23. +    FINISHED(3),
24. +    /** Status error */
```

```
25. +    ERROR(4),
26. +    /** Status aborted */
27. +    ABORTED(5),
28. +    /** All status */
29. +    ALL(99);
30. +
31. +    private int value;
32. +
33. +    AsyncMonitorTaskStatus(final int inputValue)
34. +    {
35. +        this.value = inputValue;
36. +    }
37. +
38. +    public int getValue()
39. +    {
40. +        return value;
41. +    }
42. +
43. +    public static AsyncMonitorTaskStatus getEnum(final int valTypei)
44. +    {
45. +        AsyncMonitorTaskStatus type = ERROR;
46. +        switch (valTypei)
47. +        {
48. +            case -1:
49. +                type = INACTIVE;
50. +                break;
51. +            case 0:
52. +                type = WAITING;
53. +                break;
54. +            case 1:
55. +                type = SPLIT;
56. +                break;
57. +            case 2:
58. +                type = WORKING;
59. +                break;
60. +            case 3:
61. +                type = FINISHED;
62. +                break;
63. +            case 4:
64. +                type = ERROR;
65. +                break;
66. +            case 5:
67. +                type = ABORTED;
68. +                break;
69. +            case 99:
70. +                type = ALL;
71. +        }
72. +        return type;
73. +    }
74. +
75. +}
```

Objeto AsyncMonitorDisplay :

```
1.  /**
2.  + * AsyncMonitorDisplay.java <br>
3.  + * <br>
4.  + * Description : TODO <TO BE FILLED>
5.  + * <br>
6.  + * <br>
7.  + * Modifications history :<br>
8.  + * <br>
9.  + * TODO <DD-MMM-YYYY> Creation <br>
10. + * <br>
```

```
11. + */
12. +public class AsyncMonitorDisplay extends TPMBusinessObject
13. +{
14. +    /**
15. +     * 1: start management of serialVersionUID
16. +     */
17. +    public final static long serialVersionUID = 1L;
18. +    private boolean                  task             = false;
19. +    private boolean                  subtask          = false;
20. +    private boolean                  process          = false;
21. +
22. +    /**
23. +     * Constructor
24. +     */
25. +    public AsyncMonitorDisplay()
26. +    {
27. +        super();
28. +    }
29. +
30. +    public AsyncMonitorDisplay(String title, boolean process, boolean task, boolean
     subtask)
31. +    {
32. +        super(title);
33. +        this.process = process;
34. +        this.task = task;
35. +        this.subtask = subtask;
36. +    }
37. +
38. +    @Override
39. +    public String getIdentificator()
40. +    {
41. +        return this.getTitle();
42. +    }
43. +
44. +    public boolean isTask()
45. +    {
46. +        return task;
47. +    }
48. +
49. +    public void setTask(boolean task)
50. +    {
51. +        this.task = task;
52. +    }
53. +
54. +    public boolean isSubtask()
55. +    {
56. +        return subtask;
57. +    }
58. +
59. +    public void setSubtask(boolean subtask)
60. +    {
61. +        this.subtask = subtask;
62. +    }
63. +
64. +    public boolean isProcess()
65. +    {
66. +        return process;
67. +    }
68. +
69. +    public void setProcess(boolean process)
70. +    {
71. +        this.process = process;
72. +    }
73. +
74. +    /**
75. +     * @param selectedValue
```

```
76. +     */
77. +
78. +}
```

Objeto AsyncMonitorProcess

```
1.  public class AsyncMonitorProcess implements Serializable
2.  +{
3.  +    /**
4.  +     *
5.  +     */
6.  +    private static final long          serialVersionUID = 1L;
7.  +    private String                     server_names;
8.  +    private long                       server_portl;
9.  +    private String                     client_names;
10. +    private TPMDate                    heartbeat_datel;
11. +    private AsyncMonitorProcessStatus status;
12. +    private short                      complexityi;
13. +    private boolean                    use_preallocationb;
14. +
15. +    /**
16. +     *
17. +     */
18. +    public AsyncMonitorProcess()
19. +    {
20. +    }
21. +
22. +    public String getServer_names()
23. +    {
24. +        return server_names;
25. +    }
26. +
27. +    public void setServer_names(String server_names)
28. +    {
29. +        this.server_names = server_names;
30. +    }
31. +
32. +    public long getServer_portl()
33. +    {
34. +        return server_portl;
35. +    }
36. +
37. +    public void setServer_portl(long server_portl)
38. +    {
39. +        this.server_portl = server_portl;
40. +    }
41. +
42. +    public String getClient_names()
43. +    {
44. +        return client_names;
45. +    }
46. +
47. +    public void setClient_names(String client_names)
48. +    {
49. +        this.client_names = client_names;
50. +    }
51. +
52. +    public TPMDate getHeartbeat_datel()
53. +    {
54. +        return heartbeat_datel;
55. +    }
56. +
57. +    public void setHeartbeat_datel(TPMDate heartbeat_datel)
58. +    {
```

```
59. +        this.heartbeat_datel = heartbeat_datel;
60. +    }
61. +
62. +    public AsyncMonitorProcessStatus getStatus()
63. +    {
64. +        return status;
65. +    }
66. +
67. +    public void setStatus(AsyncMonitorProcessStatus status)
68. +    {
69. +        this.status = status;
70. +    }
71. +
72. +    public short getComplexityi()
73. +    {
74. +        return complexityi;
75. +    }
76. +
77. +    public void setComplexityi(short complexityi)
78. +    {
79. +        this.complexityi = complexityi;
80. +    }
81. +
82. +    public boolean getUse_preallocationb()
83. +    {
84. +        return use_preallocationb;
85. +    }
86. +
87. +    public void setUse_preallocationb(boolean use_preallocationb)
88. +    {
89. +        this.use_preallocationb = use_preallocationb;
90. +    }
91. +
92. +}
```

Objeto AsyncMonitorProcessRequest:

```
1.  **
2.  + * AsyncMonitorProcessRequest.java <br>
3.  + * <br>
4.  + * Description : TODO <TO BE FILLED>
5.  + * <br>
6.  + * <br>
7.  + * Modifications history :<br>
8.  + * <br>
9.  + * TODO <DD-MMM-YYYY> Creation <br>
10. + * <br>
11. + */
12. +public class AsyncMonitorProcessRequest implements TPMDataRequest, Serializable
13. +{
14. +    private AsyncMonitorProcessStatus status;
15. +    private long                      last_heartbeat;
16. +
17. +    /**
18. +     *
19. +     */
20. +    public AsyncMonitorProcessRequest()
21. +    {
22. +
23. +    }
24. +
25. +    @Override
26. +    public boolean validate(TPMUserInfo userInfo)
27. +    {
```

```
28. +         return false;
29. +     }
30. +
31. +     @Override
32. +     public String toString(TPMUserInfo userInfo)
33. +     {
34. +         return null;
35. +     }
36. +
37. +     public AsyncMonitorProcessStatus getStatus()
38. +     {
39. +         return status;
40. +     }
41. +
42. +     public void setStatus(AsyncMonitorProcessStatus status)
43. +     {
44. +         this.status = status;
45. +     }
46. +
47. +     public long getLast_heartbeat()
48. +     {
49. +         return last_heartbeat;
50. +     }
51. +
52. +     public void setLast_heartbeat(long last_heartbeat)
53. +     {
54. +         this.last_heartbeat = last_heartbeat;
55. +     }
56. +
57. +}
```