ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO EN INFORMÁTICA

Título del proyecto:

INTEROPERABILIDAD Y MODERNIZACIÓN DE APLICACIONES EN ENTORNO MAINFRAME

Alumno: Mikel Castillejo Erviti

Tutor: Federico Fariña Figueredo

Pamplona, 25 de Febrero de 2011

Objeto y desarrollo

Con este proyecto se trata de mejorar la interoperabilidad y modernizar las aplicaciones CICS que operan sobre un mainframe System z de IBM. En concreto, vamos a utilizar un mainframe IBM System z10 Business Class. En general, cuando nos refiramos a mainframe nos estaremos refiriendo al mainfarme que vamos a utilizar.

Lo que vamos a buscar es integrar la lógica CICS dentro de otros sistemas de información externos al mainframe como pueden ser aplicaciones Java, .NET, etc. y "modernizar" en la medida de lo posible la interfaz que nos permite interactuar con esta lógica.

El CICS o Customer Information Control System, parte fundamental de las mainframes System z de IBM, es un gestor transaccional o monitor de teleproceso que permite procesar transacciones tanto en forma online como batch. Las formas de acceder y operar con el CICS son:

- por medio de una terminal 3270, conocidas de manera informal, debido al color del texto en los modelos originales, como terminales de pantalla verde. Como podemos ver en la Figura 0.1, este tipo de terminales ni siquiera tienen ratón.

Figura 0.1 Terminal 3270

- por medio de un emulador Telnet3270 o TN3270 funcionando sobre un sistema operativo como puede ser Windows, Linux, etc. que tenga conexión con el mainframe. Esta es la forma de acceder más habitual hoy en día.

En ambos casos la interfaz es textual, es decir, la interfaz sólo usa texto para su funcionamiento, ya sea para dibujar ventanas, menús, etc. como para introducir datos y comunicarse con la interfaz. No es posible la utilización del ratón para pinchar en un menú y que se despliegue el menú o se ejecute una acción. Esta característica hace muy poco amigable su uso. Por ejemplo:

- La Figura 0.2 muestra un ejemplo de pantalla TN3270 con un formulario de entrada de datos. Se aprecia que sólo se usa texto para "dibujar" las pantallas en vez de elementos propios de las interfaces gráficas. Podemos apreciar que los lugares donde podemos introducir datos están subrayados en lugar de tener, por ejemplo, un cuadro de texto como tendríamos en una interfaz gráfica. Esto complica al usuario la labor de localizar las zonas de la pantalla dónde debe escribir. Además, dependiendo del emulador utilizado, ni siquiera se tiene la posibilidad de acceder a dichas zonas utilizando una herramienta intuitiva como el ratón, debe acceder empleando múltiples veces el tabulador o la tecla de salto de línea.

Figura 0.2 Pantalla TN3270 con formulario de entrada

- La Figura 0.3 muestra otro ejemplo. En este caso se trata de un menú de personalización. La interfaz es poco clara: cada opción se debe seleccionar poniendo el símbolo "/" delante en lugar de utilizar un check box para seleccionar las opciones que deseemos.

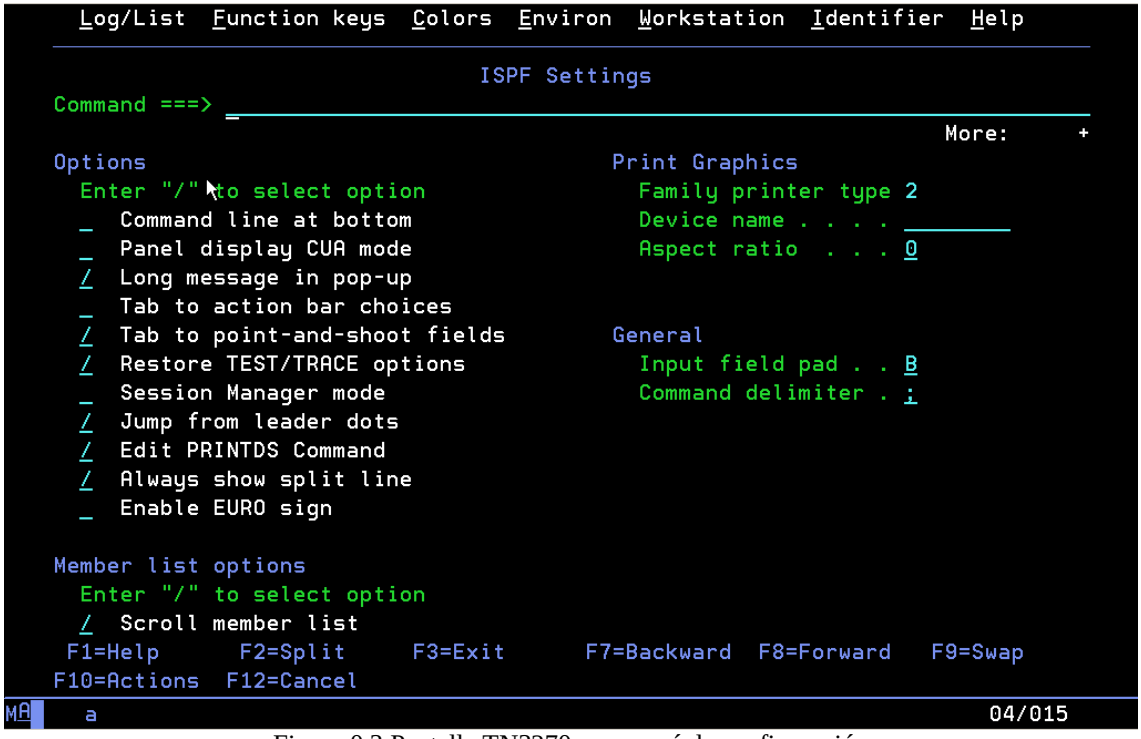


Figura 0.3 Pantalla TN3270 con menú de configuración

- La Figura 4 muestra un menú desplegable. Para desplegarlo hemos tenido que situar el cursor debajo de la letras del menú que queremos seleccionar y pulsar “ENTER” en vez de hacer clic con el ratón como haríamos en una interfaz gráfica.

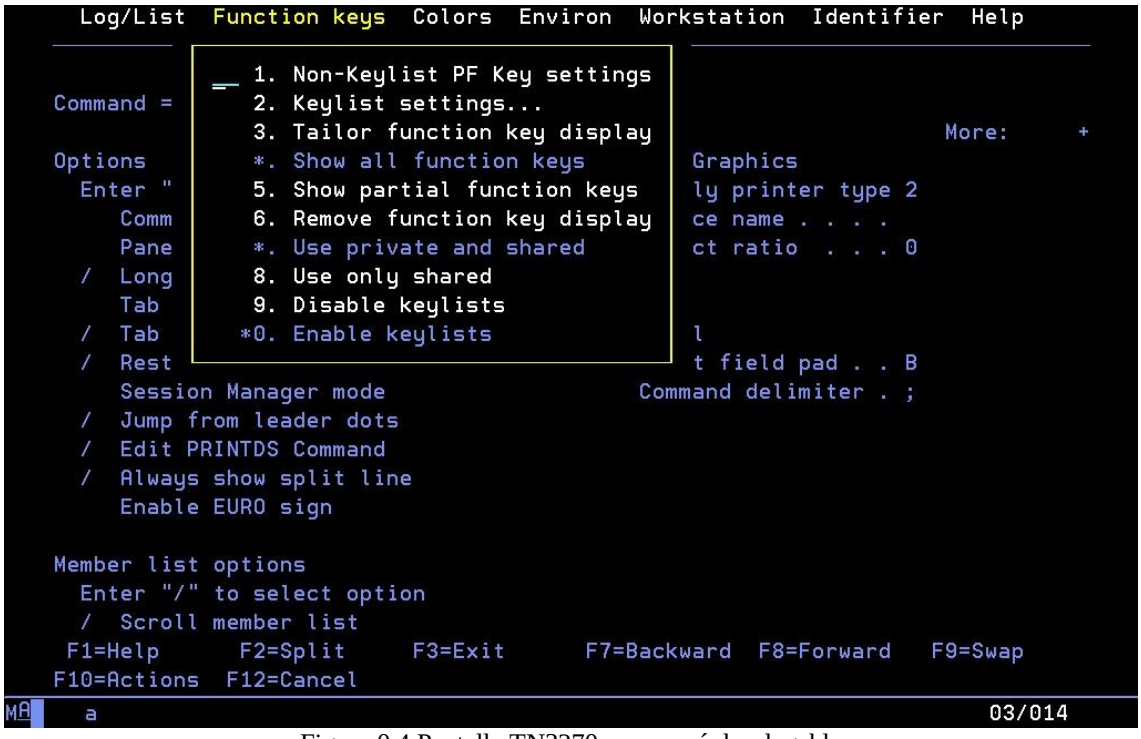


Figura 0.4 Pantalla TN3270 con menú desplegable

Algunos simuladores de TN3270, como el que utilizaremos nosotros, permiten usar el ratón pero solo para situar el cursor en el lugar que queramos. Para ejecutar la acción que queramos llevar a cabo debemos aún pulsar “ENTER”.

Tampoco es posible usar el botón derecho del ratón para acceder a diferentes opciones como copiar, pegar, etc. Para ello, y en caso de que lo permita el emulador que usemos, podemos configurar atajos de teclado. El problema es que dichas opciones las debe configurar cada usuario, no suelen venir configuradas por defecto en los diferentes emuladores.

Es obvio que el uso de un interfaz de tipo texto, aunque pueda ofrecer un buen rendimiento y productividad para usuarios expertos, es difícil de usar para usuarios principiantes. En estos usuario provoca, además, un rechazo inicial por el aspecto "anticuado" y poco amigable de las pantallas que se nos muestran. Uno de los objetivos de este proyecto consiste en mejorar este modo de trabajo de modo que los usuarios dispongan de las funcionalidades que hoy en día son comunes a todos las interfaces gráficas por medio del ratón y los interfaces gráficos.

La mejora de los interfaces se puede traducir en una mayor independencia del trabajador respecto al puesto de trabajo. Cada usuario puede tener configurado el emulador de diferente manera a fin de obtener las funcionalidades que su terminal no le proporciona. Esto puede hacer difícil que un usuario trabaje sobre el emulador de otro para hacer tareas puntuales. La introducción de interfaces más modernas y amigables (escritas en Java, .Net, etc.) haría innecesaria la personalización de los terminales.

Por otra parte, normalmente no es posible integrar el tráfico TN3270 que se genera entre el mainframe en general, y el CICS en particular, con un emulador de terminal con otros sistemas de información externos al mainframe, como pueden ser aplicaciones escritas en Java o .NET, de forma que estas aplicaciones puedan trabajar con los datos recibidos al hacer peticiones al CICS. Esto proyecto también busca integrar la lógica CICS con otros sistemas externos, en concreto con aplicaciones Java y .NET, de forma que puedan interoperar entre ellos.

A continuación vamos a ver la estructura del presente documento:

- Objeto y desarrollo
  - Muestra qué se pretende conseguir con el proyecto y la estructura del presente documento.
- Capítulo 1: Introducción
  - Explica brevemente los diferentes elementos que se van a utilizar en la realización del proyecto como qué es un mainframe, el CICS, etc.
  - Presenta diferentes soluciones para alcanzar los objetivos del proyecto y elige una de ellas.
  - Lista los diferentes elementos hardware y software que se van a necesitar para realizar el proyecto.
- Capítulo 2: CTG
  - Explica las posibilidades que ofrece una de las soluciones elegidas.
  - Muestra como instalar el producto.
  - Presenta diferentes pruebas llevadas a cabo.
- Capítulo 3: CSFF
  - Explica las posibilidades que ofrece otra de las soluciones elegidas.
  - Muestra como instalar el producto.
  - Presenta diferentes prueba llevadas a cabo.
- Capítulo 4: Conclusiones
  - Resume los logros conseguidos con este proyecto.
  - Presenta posibles vías de trabajo futuras.

1.- Introducción

1.1.- Introducción a los elemenos que vamos a utilizar

1.1.1.- Introducción a los mainframe

1.1.1.1.- Breve historia de los mainframe

Un mainframe, o computadora central, es un ordenador grande y potente usado principalmente para el procesamiento de grandes cantidades de datos como puede ser el procesamiento de transacciones bancarias. Tradicionalmente, un mainframe ha sido un ordenador pensado para el trabajo de proceso por lotes o BATCH. Este sistema se caracteriza en el tratamiento de grandes volúmenes de información, teniendo una entrada de datos, un proceso interno, y una salida de datos.

Un típico proceso BATCH es, por ejemplo, el cierre diario de cuentas en un banco donde hay que actualizar el saldo de las cuentas con las operaciones que se han ejecutado en el día. El sistema va leyendo los ficheros de cuentas bancarias y los fichero de movimientos del día, estos son los datos de entrada, los procesa para actualizar el saldo y plasmar esos movimientos en la cuenta, proceso interno, y devuelve los ficheros de cuentas actualizados y, en algunos casos, los extractos informativos que se envían a las casa, salida de datos.

Donde decimos el cierre de cuentas de un banco podemos decir el procesamiento de las nóminas de una empresa, la contratación de unas vacaciones o casi cualquier cosa que se nos ocurra. En general, un proceso BATCH es un proceso repetitivo que se realiza con millones de fragmentos de información.

Nos hay que confundir con los llamados supercomputadores. Este tipo de ordenadores se centra en problemas limitados por la velocidad de cálculo mientras que los mainframe se centran en problemas limitados por los dispositivos de E/S y la fiabilidad. El primero realiza cálculos y operaciones muy complejas sobre una cantidad “reducida” de datos mientras que el segundo realiza operaciones más sencillas sobre miles o millones de datos. Normalmente, un supercomputador es usado en áreas científicas y militares para realizar cálculos muy complejos

mientras que un mainframe es utilizado por empresas que necesitan procesar grandes cantidades de datos como por ejemplo instituciones bancarias para procesar las transacciones que se han realizado a lo largo de un día.

Entre los diferentes tipos de mainframes nos encontramos con los diferentes modelos de System z de IBM, que es el nombre que desde 2006 IBM le da a sus diferentes modelos de mainframe. Es sobre uno de estos modelos, el System z10, es sobre el que vamos a trabajar.

En 1964, y con la llegada del transistor, IBM creo el primer mainframe de la historia, el Mainframe IBM System/360. Estosprimeros mainframe, aunque potentes, eran especialmente caros.

A partir de esa fecha, y con la llegada el microprocesador, los costes se redujeron muchísimo y los mainframes empezaron a comercializarse a nivel de empresa, donde todos los procesos de producción Batch como nóminas, control de almacenes, etc. se volvieron muy eficaces. Esto hizo que se ahorraron grandes sumas de dinero y tiempo en cosas que antes se hacían de manera manual por lo que las empresas pagaban el gran precio que suponía uno de estos ordenadores. De esta forma ahorraban muchos miles de millones al año que antes invertían en realizar estas tareas de forma manual.

En la década de los 70 y 80 aparecieron varios competidores a IBM en el terreno de los mainframe como Hitachi, Fujitsu, etc. que vendían diferentes tipos de mainframe aunque todos estaban basados en la arquitectura de los mainframe de IBM. Es por ello que a pesar de que cada uno tenía su propio sistema operativo, multiprogramación, etc. todos funcionaban prácticamente de la misma manera.

En los años 70 ya era célebre el proceso interactivo donde miles de usuarios podían interactuar al mismo tiempo con el mainframe dando un buen tiempo de respuesta, realizando miles y miles de transacciones por segundo, consultas concurrentes a enormes bases de datos, etc. debido a que el sistema operativo era una pieza de gran precisión y eficacia. El único pero era el altísimo coste de los terminales, entre 12.000 y 20.000 euros de aquella época, que hacía que el tener terminales fuera un lujo demasiado caro.

Sin embargo, en la década de los 80 estas máquinas bajaron su precio hasta los 2 o 3 millones de euros y, junto a la reducción también del precio de los terminales, hicieron que miles de empresas adoptaron esa forma de trabajar.

En la década de los 90 IBM redujo bastante los precios para hacer frente al fenómeno Downsizing, un intento de descentralizar en arquitecturas más baratas toda la lógica del negocio, por lo que varios competidores abandonaron la producción de mainframes al no poder hacer frente a los precios que ofrecía IBM. Finalmente, los mainframe de IBM también se impusieron al fenómeno Downsizing debido a su gran robustez y eficiencia.

Actualmente podemos encontrarnos con la posibilidad de tener un mainframe por un millón de euros, Con esto nos encontramos con una máquina muy potente, eficaz, con gran capacidad de almacenamiento, etc. lo que la hace muy apetecible para medianas-grandes empresas que diariamente tengan que realizar operaciones con miles o millones de datos.

1.1.1.2.- Hardware y software de un mainframe

Actualmente, un modelo de mainframe de la serie System z es un armario de aproximadamente 2 metros de alto y casi otros tantos de ancho y largo, con un peso aproximado de 900 kilos, donde se sitúa los componentes principales de la máquina como lo que sería la CPU de un PC tradicional, su memoria RAM, el módulo que se encarga de gestionar el sistema de I/O, etc.

En diferentes armarios situados aparte, y aproximadamente del mismo tamaño, nos encontramos con los distintos periféricos de E/S como pueden ser las unidades de cinta, de disco, etc.

Estos módulos pueden tener su propio sistema operativo almacenado en firmware o en microcódigo que dotan al módulo de una cierta inteligencia.

Actualmente las unidades de cinta de un mainframe son cartuchos de 300GB aunque se están empezando a comercializar unidades de 600GB y hasta de 1,2TB que, a un precio que no llega a los 60 euros, hace que esta forma de almacenamiento sea la más barata para almacenar grandes cantidades de datos, aunque también es la más lenta a la hora de acceder a la información.

Por último, tendríamos diferentes conexiones externas utilizadas para operar con el mainframe. Entre estas nos podemos encontrar las terminales 3270 propiamente dichas, similares a la que hemos visto anteriormente, o un PC normal corriendo un emulador de 3270, un servidor de aplicaciones que realiza diferentes peticiones al mainframe, etc.

En cuanto al software, el sistema operativo que utilizan estas máquinas también ha sufrido cambios desde sus inicios.

Cuando IBM lanzó sus primeros modelos de mainframe, en 1964, desarrolló dos sistemas operativos llamados DOS y OS/360.

El primero de ellos estaba pensado para máquinas pequeñas mientras que el segundo era el sistema operativo elegido en los mainframe más grandes y potentes. Nos vamos a fijar en este último ya que es su evolución la que ha dado lugar al sistema operativo z/OS que es el que vamos a utilizar.

De este sistema operativo aparecieron varias versiones llamadas PCP, MFT y MVT:

- El primero, el PCP o Primary Control Program, era similar a DOS en el sentido de que ambos eran monotarea, aunque necesitaba más recursos para funcionar, por lo que prácticamente no llegó a salir de los laboratorios de IBM.
- El MFT, Multiprogramming with Fixed number of Tasks, Multipogramación con número Fijo de Tareas, salió en 1966, meses más tarde que el PCP, y fue el primer sistema operativo multitarea de la historia. Tenía la pega de que debías saber previamente cuanto iba a ocupar una ejecución de un proceso ya que si ocupaba más que la región seleccionada no se ejecutaba.
- El MVT, Multiprogramming with Variable number of Tasks, Multipogramación con número Variable de Tareas, era similar al MFT pero podía crear y borrar regiones de memoria sin tener que apagar todo el sistema por lo que no necesitabas conocer a priori cuanto iba a ocupar un proceso. A partir de esta versión evolucionó el sistema operativo z/OS actual.

Más adelante, al OS/360 se le añadió el TSO, o Time Sharing Option, que permitía a cientos de terminales trabajar simultáneamente contra la misma máquina.

Del MVT se pasó al MVS, Multiple Virtual Storage, o Múltiple Almacenamiento Virtual, en 1974 cuando le añadieron múltiples espacios de direcciones virtuales. Este sistema operativo se renombró a MVS/370 con la llegada de los nuevos mainframes de IBM. Más tarde se renombró a MVS/XA y MSV/ESA por los nombres en inglés de arquitectura extendida y arquitectura de sistemas empresariales respectivamente.

Con la llegada de los nuevos modelos z/390 y la compatibilidad con Unix se paso a llamar OS/390 en 1995 y finalmente con la llegada de los 64 bits y el cambio nombre de los mainframe a zSeries actualmente tenemos el z/OS desde el año 2000.

En la Figura 1.1 podemos ver, por décadas, la evolución del sistema operativo z/OS, el que opera actualmente en los mainframe, junto con la inclusión de ciertos componentes al mismo.

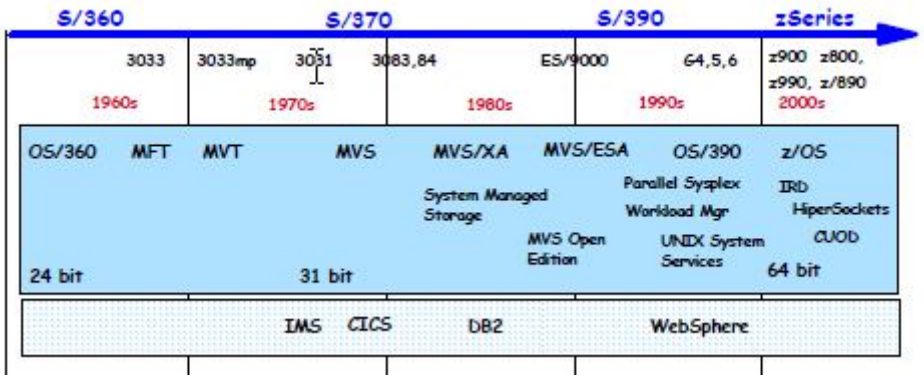


Figura 1.1 Evolución del sistemas operativos z/OS IBM para mainframes

En la Figura 1.2 podemos ver la evolución del sistema operativo junto a la evolución de la arquitectura.

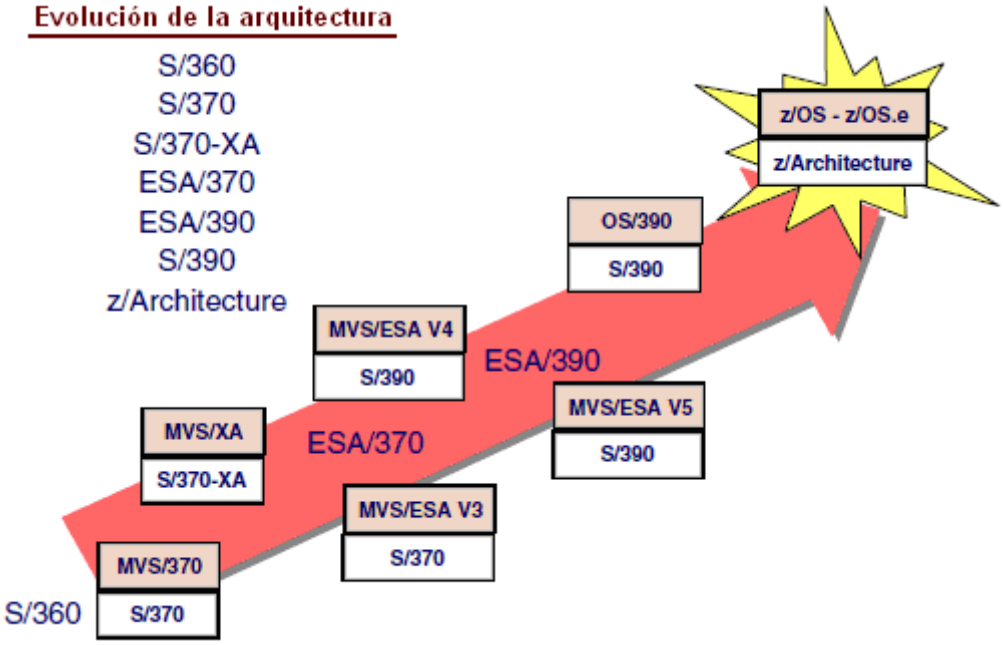


Figura 1.2 Evolución del sistema operativo z/OS y de la arquitectura

Aunque z/OS ofrece muchas de las funcionalidades de los sistemas operativos modernos también nos sigue ofreciendo muchas de las funcionalidades que ofrecía en sus orígenes, o en evoluciones anteriores del sistema operativos, como el soporte para COBOL, y el CICS, RACF, DB2, etc. que explicaremos más adelante.

El actual sistema operativo z/OS soporta las APIs de Unix al tener una parte de compatibilidad con servicios Unix.

También nos ofrece la posibilidad de tener en una misma máquina diferentes particiones lógicas o LPARs, de Logical PARtition, similar a las particiones que nos podemos encontrar en otros sistemas operativos. Esto nos ofrece la posibilidad de tener virtualmente diferentes máquinas independientes sobre un mismo mainframe, cada una de ellas ejecutando su propio sistema operativo. Cada LPAR puede compartir o tener de forma dedicada diferentes recursos hardware como procesadores, dispositivos de E/S, etc. Esto nos permite por ejemplo, teniendo una única máquina, hacer pruebas en una LPAR si riesgo a que un posible fallo ponga en peligro todo el sistema, tener diferentes configuraciones del sistema para diferentes propósitos, etc.

Una vez que hemos vista la evolución que ha sufrido el sistema operativo de los mainframe vamos a ver diferentes módulos o subsistemas que habitualmente nos podemos encontrar en un mainframe. Estos subsistemas son los siguientes:

- TSO
- JES2
- SMP/E
- VTAM
- RACF
- Servicios UNIX de z/OS
- CICS
- DB2

El TSO, o Time Sharing Option, es un subsistema que nos permite accede al mainframe para trabajar con él. Es decir, por medio de un PC corriendo un emulador de terminal 3270, o de una terminal 3270 en caso de tenerla, puedo hacer login en una sesión de TSO. De esta forma puedo controlar cualquier elemento de la máquina que se quiera como ejecutar un programa, escribir, copiar, borrar, etc. un data set, similar a un fichero o archivo en un sistema operativo "tradicional", etc. Es decir, es similar en un PC corriente a estar sentado delante de la pantalla realizando las operaciones que se consideren oportunas.

El JES2, o Job Entry Subsystem versión 2, es el planificador de tareas del sistema. Tenemos un lenguaje de programación llamado JCL, Job Control Languaje, con el que podemos escribir programas tipo bat en MS-DOS a los que se les llama Job. Estos Jobs los podemos submitir, o ejecutar, de forma que entra en una cola de entrada del JES2 para su ejecución. El JES2 se encarga de gestionar todos aquellos recursos que necesitan los diferentes Jobs para ejecutarse sin que se produzcan bloqueos. Una vez terminada su ejecución almacena el resultado de ejecutar el Job en una cola de salida.

El JES2 también controla programas cargados en memoria en forma de tareas que se puede inicializar en tiempo de arranque del sistema operativo, o más adelante en el momento que consideremos oportuno. Este tipo de tareas se llaman *Started Tasks*.

Por último, un usuario conectado por medio de TSO será un TSU o Time Sharing User.

Para el sistema operativo todos estos son usuarios, es decir, alguien o algo que necesita y obtiene recursos del sistema operativo. En función del tipo de usuario que sea, el JES2 lo tratará de diferente manera.

El SMP/E, o System Modification Program/Extended, es una utilidad del sistema operativo z/OS, y también de sus predecesores. Nos permite instalar nuevos productos en nuestro mainframe, así como actualizar y aplicar parches a productos que ya tenemos instalados. También nos permite llevar un inventario de los productos instalados en nuestro sistema y de sus versiones, así como volver a versiones anteriores de un producto en caso de que las versiones instaladas nos den problemas.

El subsistema conocido como VTAM, o Virtual Telecommunication Access Mode, nos proporciona la capa de red necesaria para la comunicación inter-procesos, comunicación con el exterior, etc. Originalmente funcionaba usando el protocolo SNA pero actualmente también puede hacer uso del protocolo TCP/IP. Al final, lo que hace esta segunda opción es encapsular el tráfico SNA dentro del TCP/IP.

El RACF, Resource Access Control Facility, es el subsistema que se encarga de proveer la capa de seguridad del sistema. Se encarga de dar acceso a los usuarios, dar permisos sobre los diferentes recursos del sistema, etc. También se encarga de definir qué programas pueden obtener recursos de otros programas, que ficheros pueden usar, etc.

Los servicios UNIX de z/OS proveen al sistema operativo de un sistema operativo UNIX que algunos productos necesitan para funcionar. En nuestro caso, un producto lo va a necesitar para almacenar ciertos archivos de configuración mientras que otro lo va a necesitar para almacenar ciertos archivos de propiedades de los programas que desarrollemos.

Por último, vamos a ver dos programas que no forman parte propiamente dicha del sistema operativo pero que si nos los podemos encontrar instalados en la mayoría de los mainframe.

El CICS, o Customer Interface Control System, es un gestor transaccional o monitor de teleproceso que permite procesar transacciones tanto en forma online como batch. Este programa se arranca como una *Started Task* dentro del JES2.

Diferentes usos de CICS nos los encontramos en las transacciones utilizadas en un banco para consultar en saldo de una cuenta o cuando vamos a reservar unas vacaciones en una agencia de viajes, donde la pantalla de la agencia de viajes normalmente trabaja contra una aplicación llamada Amadeus que está corriendo bajo CICS.

Por otra parte nos encontramos con DB2, o DataBase 2, que es una robusta base de datos transaccional donde se almacenan la mayoría de los datos de un mainframe, lo que es similar a decir que almacena la mayoría de los datos “importantes” del mundo. Tiene multitud de herramientas para gestionarla, realizar consultas, etc. y también se arranca como una *Started Task* en el JES2.

1.1.1.3.- Data sets

Una parte importante del sistema operativo de un mainframe son los data sets. Un data set, o conjunto de datos, es un archivo de un ordenador que tiene organización en forma de registros. El término pertenece a los sistemas operativos de los mainframes de IBM y se viene usando desde el OS/360 hasta el actual z/OS.

Dependiendo de cómo están organizados los datos de un data set nos encontramos con:

- data set secuencial (physically sequential o PS)
- data set indexado (indexed sequential o IS)
- data set particionado (partitioned o PO)
- data set de acceso directo (direct access o DA).

Podríamos pensar en un PS como en un fichero y en un PO como en un árbol de directorios pero ahí acaba la similitud con el sistema de archivos de un sistema operativo Windows o Unix. Estos dos tipos de data sets son los más utilizados.

Un data set particionado (partitioned data set o PDS) es aquel que contiene *miembros* dentro de él, cada uno a su vez conteniendo un conjunto de datos independientes entre sí. Como ya hemos dicho, esto es similar a un conjunto de directorios en otro sistema operativo. A este tipo de data sets también se les llama librerías ya que a menudo se utilizan para almacenar los programas ejecutables o librerías con los códigos fuente de los programas.

Cada miembro de un PDS puede ser accedido de forma directa usando la estructura del data set. Una vez que hemos localizado el miembro que queremos se puede manejar de la misma manera que un PS.

Cuando borramos miembros de un PDS este espacio queda sin utilizar. Lo mismo ocurre cuando modificamos un miembro ya que esto causa que automáticamente se almacene en otro sitio, dejando también ese espacio sin utilizar. Para que ese espacio se vuelva a utilizar hay que comprimir el data set. Además, cuando se llena el espacio asignado a un PDS, no podemos ampliarlo y hay que usar un nuevo PDS.
Para solucionar estos problemas apareció el PDSE o PDS extendido. Además de solucionar estos problemas también cuenta con un índice para realizar de forma más rápida las búsquedas de un nombre de un miembro.

Un data set secuencial (sequential data set o SDS) es aquel en el que la información está grabada de forma secuencial y para llegar a un cierto dato del data set tenemos que recorrer todos los datos anteriores por lo que se puede hacer la similitud con un fichero.

En cuanto al nombre de un data set, hay varias cosas que tenemos que tener en cuenta. El nombre de un data set se forma por varios sub-nombres o cualificadores (qualifiers) separados por un punto. Por ejemplo, un nombre válido para un data set puede ser MIKEL.USER.FUENTES y otro MIKEL.USER.LOADLIB. Si es un PDS, al final, entre paréntesis, pondríamos el nombre del miembro, por ejemplo, MIKEL.USER.FUENTES(PROG1), para referirnos al miembro PROG1 dentro del data set MIKEL:USER:FUENTES.

Hay que tener varias cosas en cuenta al darle un nombre a un data set:

- El nombre completo, con todos sus cualificadores, tiene que ser de un máximo de 44 caracteres incluidos los puntos de separación entre cada cualificador
- Se puede usar un máximo de 22 cualificadores
- El primer cualificador se conoce como High Level Qualifier o HLQ (cualificador de alto nivel)
- El último cualificador se conoce como Low Level Qualifier o LLQ (cualificador de bajo nivel)
- Cada uno de los cualificadores tiene que ser de un máximo de 8 caracteres y empezar por un carácter alfabético o los símbolos @, $ o #
- Se pueden usar lo caracteres alfabéticos, numéricos o los símbolos @, $, # o –
- Normalmente se siguen cierta convenciones en los cualificadores
    - El HLQ de un data set de usuario está controlado por el sistema de seguridad
    - Las letras LIB indican que es una librería
    - Las letras JCL indican que contiene JCLs
    - Las letras LOAD o LINK indican que contiene ejecutables
    - Las letras PROC o PRC indican que contiene procedimientos JCL, por ejemplo procedimiento de arranque o Started Class
    - Usar muchos cualificadores se considera una mala práctica

Un tipo particular de data set es el VSAM o Virtual Storage Access Method y se refiere tanto al tipo de data set como al método de acceso. Son data sets que proporcionan funciones más complejas que otros accesos a disco y además almacenan los registros de forma que no son entendibles por otros métodos de acceso.

Estos data sets son usados por aplicaciones y no contienen programas fuente, JCLs o módulos ejecutables. Tampoco pueden ser editados o accedidos por medio de rutinas o con el ISPF que veremos más adelante.

Dependiendo de la forma en que están organizados los registros nos podemos encontrar con cuatro tipos de data sets VSAM:

- Key Sequenced Data Set o KSDS
  - Cada registro cuenta con uno o más campos de clave y los registros pueden ser accedidos o insertados por el valor de su clave
- Entry Sequenced Data Set o ESDS
  - Son registros en orden secuencial usados por IMS, DB2 o z/OS Unix
- Relative Record Data Set o RRDS
  - Permite accede a sus registros por su número aunque estos números son aleatorios. Asume que la aplicación tiene una forma de conocer los números de los registros
- Linear Data Set o LDS
  - Los datos son un stream de bytes. Es el único stream de bytes en z/OS y raramente es usado en aplicaciones

1.1.2.- Introducción a CICS

1.1.2.1.- Qué es CICS

CICS, o Customer Information Control System, Sistema de Control de Información de Cliente, es una familia de productos líder como sistema de procesamiento de transacciones online y pertenece a IBM. La familia de productos CICS engloba servidores de transacciones, conectores, como el CICS TG que veremos más adelante, herramientas de desarrollo y monitorización, etc. Aunque el nombre CICS hace referencia a una gran familia de productos, normalmente se utiliza para referirse al CICS Transaction Server para z/OS, o CICS TS.

El CICS TS, simplemente CICS a partir de ahora, es un gestor transaccional o monitor de teleproceso que se instala en computadoras de tipo mainframe de IBM que funcionan con el sistema operativo z/OS.

CICS está diseñado para procesar transacciones en modo online y batch y puede dar servicio a miles de transacciones por segundo. Normalmente el procesamiento de transacciones es interactivo con el usuario, a través de terminal, pero también es posible ejecutar transacciones en segundo plano sin la intervención del usuario.

Como dato, la revista Fortune dice que más del 90% de las compañías pertenecientes a la lista Fortune 500 utiliza CICS funcionando sobre z/OS para el procesamiento de su negocio principal.

Como gestor transaccional, CICS ofrece varias funcionalidades:

- Controla los programas en ejecución al mismo tiempo que atiende a las peticiones de los usuarios conectados online.
- Acceso a bases de datos y ficheros
- Posibilidad de conectarse a otros miembros de la familia CICS a través de SNA o de TCP/IP
- Proceso de recuperación y protección de datos en caso de producirse un problema en el sistema

Vamos a ver diferentes elementos que intervienen en la ejecución de aplicaciones dentro de CICS.

1.1.2.2.- Transacciones

Una transacción es una unidad de procesamiento que consta de uno o más programas. Se inicia por medio de una petición, a menudo desde un terminal, aunque se puede realizar desde un sistema externo, desde la ejecución de otra transacción, etc., por lo general usando el identificador de cuatro caracteres de una transacción. Su ejecución puede requerir la apertura de una o más tareas.

Además de las transacciones que podemos crear y definir, CICS provee una serie de transacciones de sistema, cuyo identificador empieza por C y que nos permiten realizar diferentes funciones. En este proyecto vamos a utilizar las siguientes:

- **CESN:** Transacción suministrada por el sistema para identificarnos en el CICS.
- **CEMT:** Permite solicitar información de los recursos CICS que se están ejecutando en el sistema y cambiar sus parámetros de control dinámicamente.
- **CEDA:** Permite definir, modificar y consultar los recursos de CICS. También permite instalar los recursos definidos para que se ejecuten en el CICS.
- **CEDF/CEDX:** Permite usar la Utilidad de Diagnóstico de Ejecución (Execution Diagnostic Faculity o EDF) para depurar programas.
- **CMAN:** Permite manejar los flujos creados con el CSFF que veremos más adelante. Permite instalar nuevos flujos, habilitarlos y deshabilitarlos, borrarlos, activar trazas, etc.

Tenemos que diferenciar dos tipos de transacciones según su tipo de ejecución:

- Transacciones conversacionales/ pseudo-conversacionales
- Transacciones no-conversacionales

1.1.2.2.1.- Transacciones conversacionales/ pseudo-conversacionales

Estas transacciones son aquellas que interaccionan con el usuario. Por ejemplo, podría ser una transacción que te muestre los diferentes artículos que hay en un almacén y que se quede esperando a que pidas detalles sobre uno de ellos para darte más información sobre él como cantidad en stock, precio de venta a clientes, etc. Tanto la entrada como la salida de datos se realizan por medio de una terminal.

La diferencia entre las transacciones conversacionales y las pseudo-conversacionales es que las primeras cargan un programa en memoria y permanece ahí hasta que finaliza su ejecución aunque no este activo, por ejemplo porque está esperando a una entrada de datos desde terminal. Las segundas cargan el programa en memoria pero realizan conexiones y desconexiones descargando el programa de memoria cuando está en espera, liberando así los recursos que está consumiendo la tarea. Este segundo tipo requiere un modo especial de codificar los programas de forma que se establezcan una serie de ciclos que permiten proseguir una tarea a partir del punto en que se produzco la última desconexión.

1.1.2.2.2.- Transacciones no-conversacionales

Este segundo tipo de transacciones son aquellas a las que se les pasa un área de datos con los parámetros que necesita para la ejecución y devuelve los resultados de la misma forma, sin que el usuario interactúe con la transacción mientras se está ejecutando. Un ejemplo sería una transacción a la que le pasamos en un área de datos un apellido y nos devuelve todos los

usuarios que hay en un sistema con ese apellido pero no nos permite, por ejemplo, interactuar con la transacción por medio de una terminal para pedir más detalles sobre un usuario.

Según el tipo de área de datos que utilicemos para comunicarnos con las transacciones nos encontramos con transacciones basadas en COMMAREA o basadas en Canales y Contenedores.

Aunque más adelante explicaremos que es cada una y sus similitudes y diferencias, básicamente una COMMAREA es un área de memoria que contiene datos pero que no posee ni nombre ni formato. Los Canales y Contenedores son áreas de memoria que contiene datos y además tienen un nombre y formato.

Por ejemplo, la información de una persona, como DNI, nombre, primer apellido, etc. en una COMMAREA estaría toda seguida. Con los Canales y Contenedores tendríamos una zona llamada DNI con la información correspondiente, otra para el nombre, primer apellido, etc.

1.1.2.3.- Programas

Un programa es el conjunto de instrucciones que se ejecutan en el CICS para llevar a cabo una determinada tarea.

Los programas en CICS pueden estar escritos en los siguientes lenguajes:

- Ensamblador
- COBOL
- PL/I
- C
- C++
- Java

En cuanto a los lenguajes utilizados, algunos estudios revelan los siguientes datos:

- El 85% de todos los programas en mainframes están escritos en Cobol
- El 7% están escritos en ensamblador, C o C++
- El 5% están escritos en PL/I
- El 3% están escritas en Java

La mayoría de las instrucciones que se van a ejecutar se expresan usando las instrucciones propias del lenguaje pero, para hacer peticiones de servicios CICS, tenemos que usar alguna de las siguientes opciones:

- Comandos EXEC CICS proporcionados por la API de programación de aplicaciones de CICS
- Biblioteca de clases de Java para CICS (JCICS)
- Biblioteca de clases de C++ para CICS

1.1.2.4.- Tareas

Una tarea es una instancia de la ejecución de una transacción. CICS. El CICS junto con el sistema operativo, maneja el control de múltiples tareas mediante la asignación de un proceso de sistema a cada tarea. Mientras que una tarea está a la espera, por ejemplo a la espera de leer un

archivo o esperando la respuesta desde una terminal, el sistema operativo puede dar el control a otra tarea.

1.1.2.5.- LUWs

La ejecución de transacciones dentro del sistema se puede agrupar en unidades lógicas de trabajo *(Logical Units of Works o LUWs)*.

Cada LUW es un conjunto de cambios asociados a los datos. Por ejemplo, en un sistema de contabilidad, una LUW consiste en la actualización de las cuentas por pagar, la actualización de los libros, y la creación de un cheque donde cada parte de la LUW podría ser una transacción.

El trabajo que realiza cada LUW es completamente independiente de la labor que realiza cualquier otra LUW. Si una LUW realiza modificaciones en múltiples recursos, todos ellos se modifican satisfactoriamente o no se modifica ninguno de ellos.

El sistema de procesamiento de transacciones se asegura de que, cuando varios usuarios tienen acceso a los recursos, los cambios parciales que se realizan en una LUW no son puestos a disposición de otros usuarios hasta que la LUW se ha completado.

Una LUW puede terminar correctamente y verificar los cambios, commit, o producirse un error en su ejecución y deshacer los cambios realizados, *backout* o rollback. Cuando una LUW realiza *commit* sus cambios son permanentes.

1.1.3.- Introducción a TSO e ISPF

1.1.3.1.- TSO

La utilidad TSO, Time Sharing Oprtion u Opción de Tiempo Compartido, es un entorno interactivo de tiempo compartido utilizado para realizar la mayoría de las tareas sobre el sistema operativo z/OS de IBM, aunque está disponible desde el sistema operativo OS/MVT. TSO primeramente se ofrecía como una opción en los primeros sistemas operativos MVT pero paso a formar parte del sistema operativo a partir de la salida del MVS en 1974 y se ha mantenido hasta nuestros días.

TSO permite a cientos o miles de usuarios tener acceso a MVS al mismo tiempo pero de forma que para cada usuario parezca que es el único usuario en el sistema.

Algunas de las tareas que podemos realizar por medio del TSO es el de crear y modificar Jobs, que luego podemos submitir, comprobar los resultados producidos por los Jobs sin tener que esperar a que se produzcan informes por escrito, gestionar data sets, etc.

Aunque TSO se puede utilizar en modo línea-por-línea donde escribimos un comando y esperamos a que nos devuelva una respuesta, el modo más común de usarlo es por medio de menús o paneles. El uso de TSO por medio de paneles se realiza a través de otra utilidad llamada ISPF y a la combinación de ambos se conoce como TSO/ISPF.

Desde cualquier panel en el que estemos, tenemos una línea de comandos desde donde podemos ejecutar comando de TSO.

TSO también se utiliza para acceder al entorno de tipo Unix que hemos comentado anteriormente. Tenemos la opción de acceder a los servicios Unix por ISH o por OMVS.

El primero, también conocido como ISHELL o ISPF Shell, proporciona una interfaz por paneles similar a la de TSO e ISPF que veremos más adelanta. Para acceder a esta interfaz podemos simplemente ejecutando desde cualquier menú o panel en el que nos encontremos el siguiente comando: tso ish

También podemos acceder a ISH por medio de los paneles con la opción OM y luego, dentro de esta, con la opción ISH.

En la Figura 1.3 podemos ver la pantalla inicial de ISH.

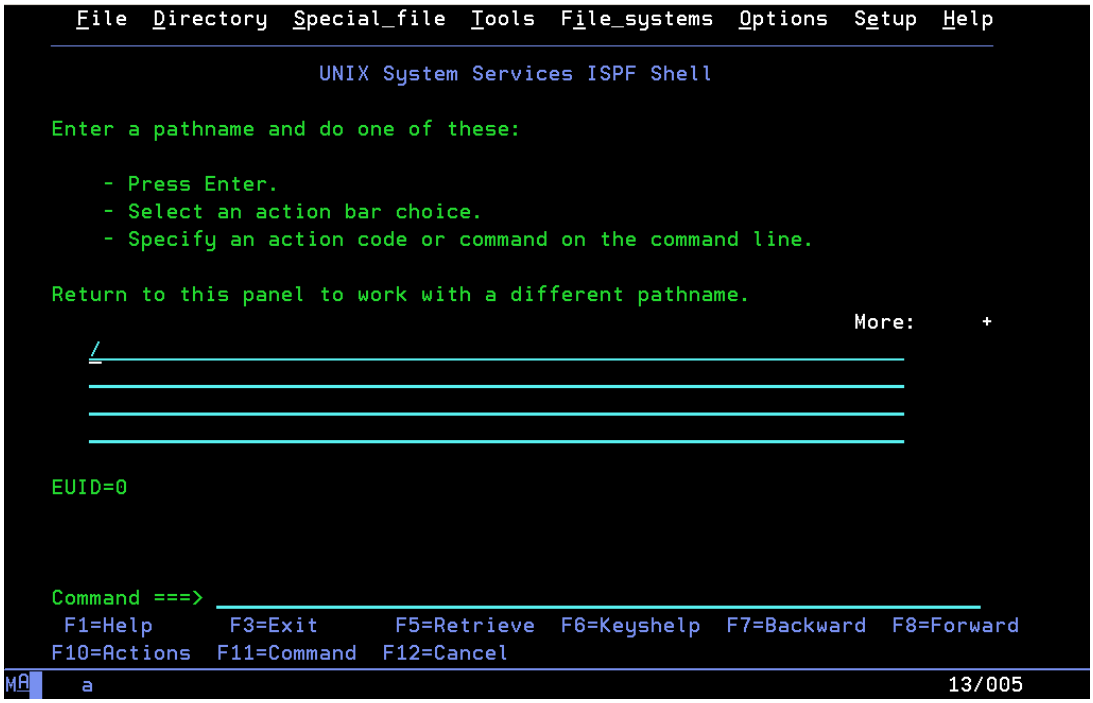


Figura 1.3 Pantalla inicial de ISH

Una vez que hayamos introducido el path al que queremos acceder, nos encontraremos con una pantalla similar a de la Figura 1.4 desde nos manejaremos por paneles de forma similar a como lo hacemos en TSO.

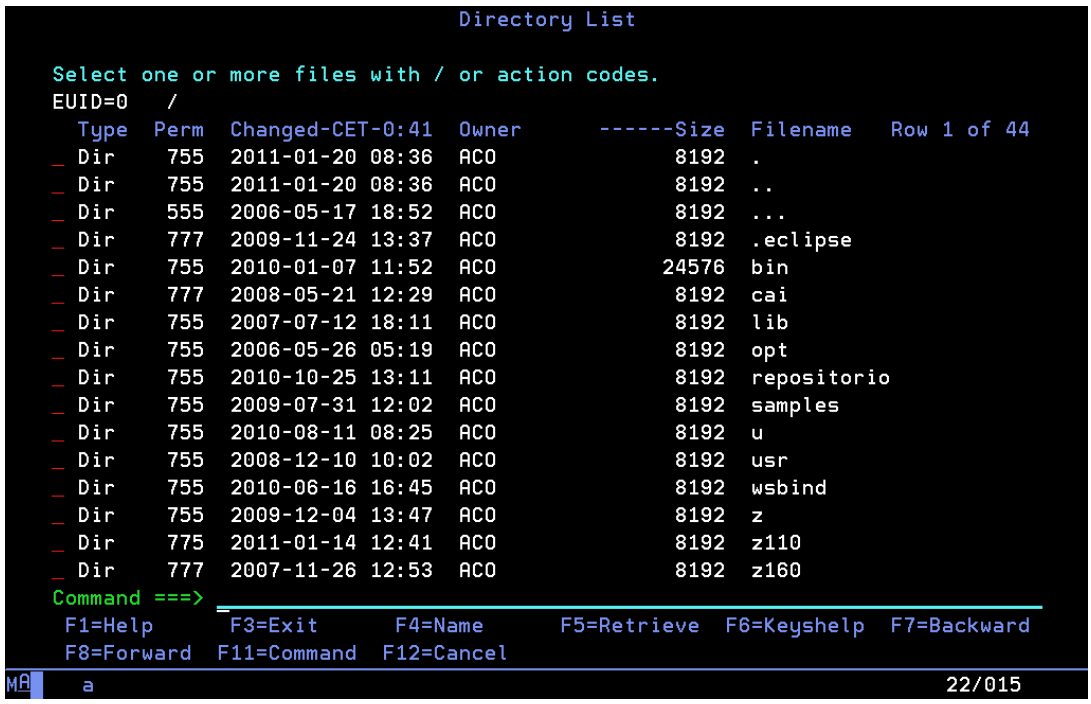


Figura 1.4 Pantalla de ISH

El segundo método proporciona una interfaz similar a la consola de comando de un sistema operativo Unix/Linux. Para acceder a este método ejecutamos desde cualquier menú el siguiente comando: tso omvs

Con ello accederemos a la pantalla mostrada en la Figura 1.5 desde nos manejaremos de forma similar a una consola de comando UNIX, incluyendo los mismos comandos, etc.

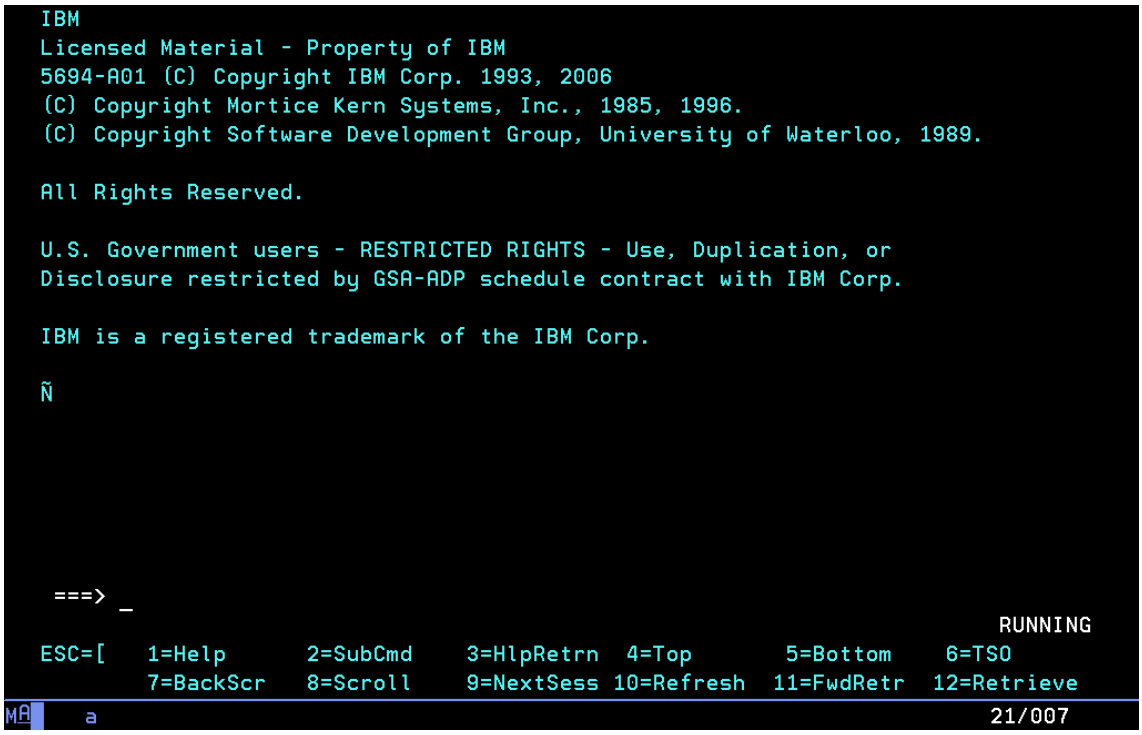


Figura 1.5 Pantalla de OMVS

1.1.3.2.- ISPF

ISPF es una aplicación que nos permite, por medio de paneles, realizar diferentes operaciones sobre TSO como por ejemplo manejar data sets, introducir comandos de TSO, etc.

Para poder hacer uso de ISPF el usuario primero se ha tenido que autentificar en el sistema por medio de TSO.

ISPF provee al usuario de un editor de textos, un navegador de archivos, así como funciones para localizar y listar data sets.

Normalmente ligado a ISPF va el PDF o Product Development Facility que es la utilidad usada para manipular data sets secuenciales o miembros de data set particionados.

Aunque frecuentemente ISPF es utilizado para manipular archivos por medio del PDF, el ISPF es ampliable y se puede utilizar como interfaz para operar con otros programas o utilidades. De hecho, muchos productos que vende IBM para MVS utilizan los paneles de ISPF para acceder a sus herramientas.

La Figura 1.6 muestra un miembro de un data set que hemos abierto por medio del ISPF/PDF y que podemos consultar, modificar, etc. En la parte superior de la pantalla podemos observar como el miembro abierto es el PREGCONG del data set D424350.USER.JCLLIB.

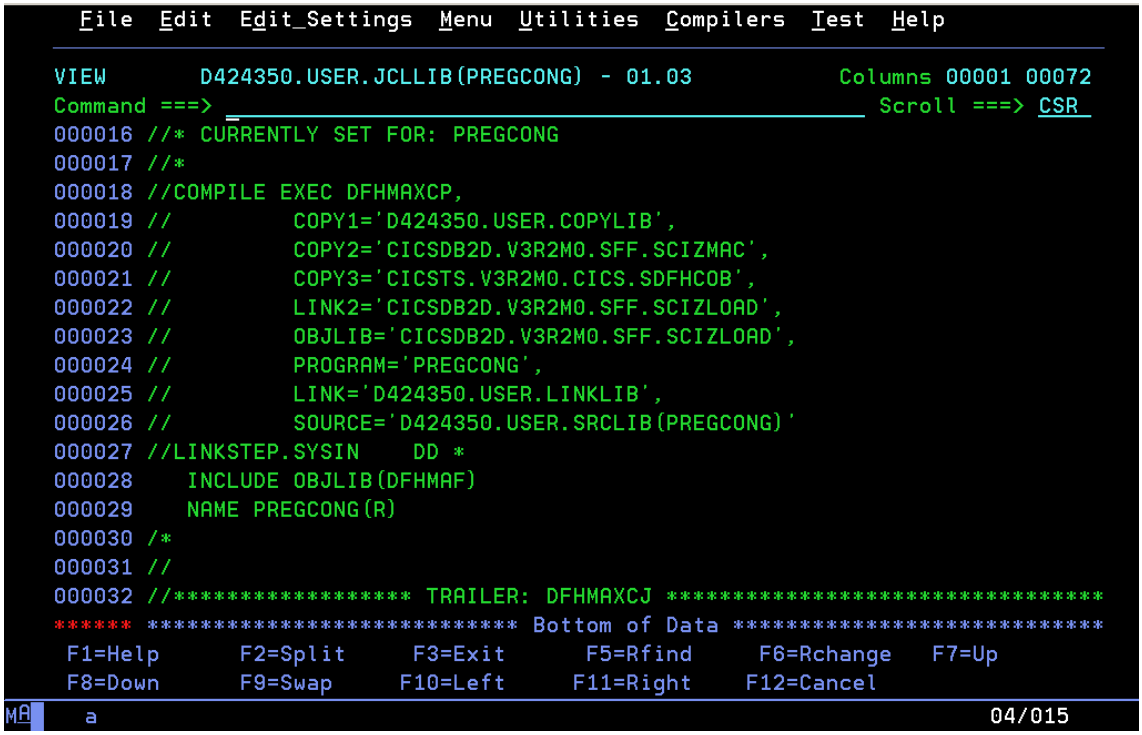


Figura 1.6 Archivo abierto con ISPF/PDF

Para poder submitir este miembro, simplemente escribimos “sub” en la línea “Command” n la parte superior de la pantalla y pulsamos Control.

En la Figura 1.6 también podemos observar, en la parte inferior, otro de los “inconvenientes” de una interfaz de tipo texto. Para desplazarnos arriba y abajo en un data set, también en un menú de opciones, un listado, etc. no podemos hacer uso de una barra de desplazamiento o de la rueda de ratón, sino que tenemos que usar teclas programadas para tal fin, normalmente F7 para ir hacia arriba y F8 para ir hacia abajo.

Los seis primeros caracteres nos indican la línea del data set en la que nos encontramos pero también nos permiten realizar tareas como copiar, borrar, etc. Por ejemplo, si quisiéramos copiar la línea 18 debajo de la línea 30 situaríamos el cursor sobre alguno de los seis primeros caracteres de la línea 18, escribiríamos c y pulsaríamos Control (el equivalente a ENTER en nuestro emulador de terminal). A continuación situaríamos el cursor en la línea 30, escribiríamos a (para que se copie la línea seleccionada a continuación de la 30) y pulsaríamos Control de nuevo.

A continuación podemos ver un listado de alguna de las operaciones que podemos realizar usando estos seis primeros caracteres de cada línea.

- **i** Insertar una línea. Cada vez que hayamos escrito algo en una línea, pulsaremos ENTER (Control en nuestro emulador) para insertar una nueva línea sin necesidad de volver a escribir i. Para salir del modo de inserción pulsaremos ENTER sin haber escrito nada.
- **i5** Insertar cinco líneas
- **d** Eliminar una línea
- **d5** Eliminar cinco líneas
- **dd/dd** Eliminar un bloque de líneas. Si queremos borrar desde la línea 6 hasta la 10, escribimos dd en la línea 6 y lo mismo en la línea 10 y pulsamos ENTER.
- **r** Repetir una línea
- **rr/rr** Repetir un bloque de líneas

- **c** Copiar una línea. Con **a** o **b** la pegamos antes o después de la línea seleccionada
- **c5** Copiar 5 líneas. Las pegamos con **a** o **b**
- **cc/cc** Copiar un bloque de líneas
- **m** Mover una línea. Al igual que con los comandos de copiar, **m5, mm/mm** y su uso con **a** o **b** son opciones válidos
- **x** Excluir una línea. No la borra, simplemente no la muestra por pantalla

1.1.3.3.- Ejemplos

A continuación vamos a poder ver unos ejemplos de TSO/ISPF. Cundo abrimos una sesión de TSO, y una vez identificados en el sistema, nos vamos a encontrar con una pantalla mostrada en la Figura 1.7.

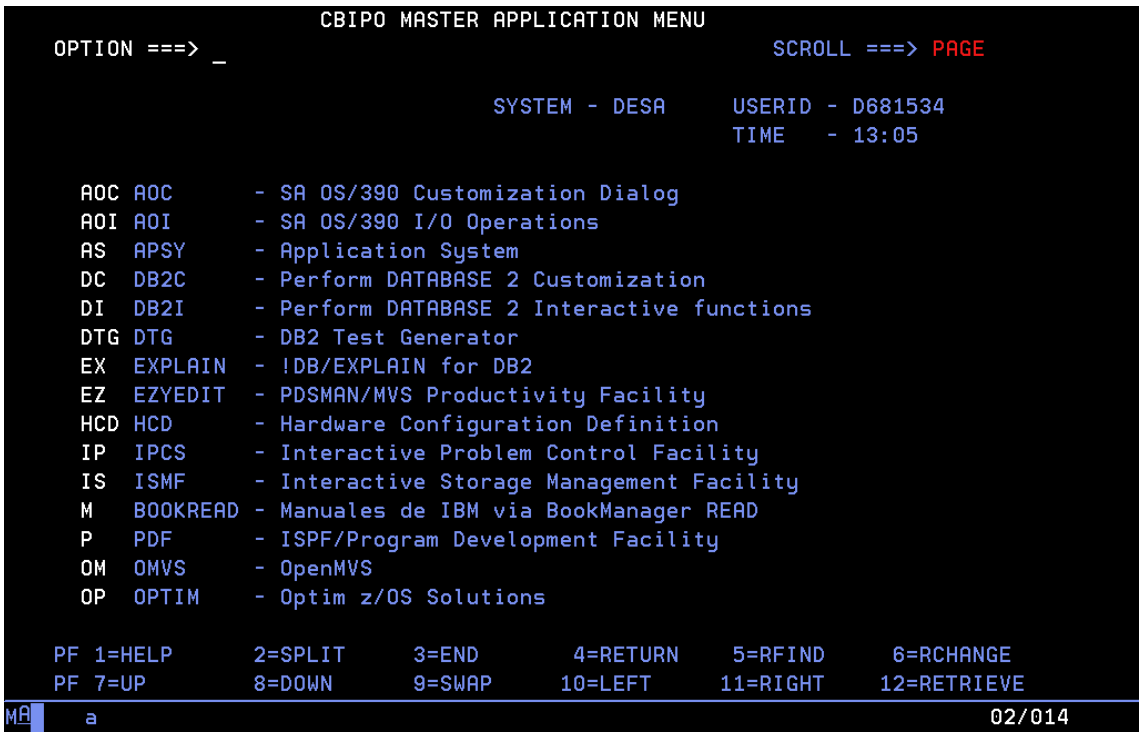


Figura 1.7 Pantalla inicial de TSO

En esta pantalla podemos ver las diferentes operaciones que podemos ejecutar. Para seleccionar una de estar operaciones bastaría con teclear los caracteres que aparecen en blanco ligados a cada una de las operaciones o utilizar algunas de las teclas PF, Programable Function Keys, o Teclas de Función Programable, normalmente las teclas F1-F12, para realizar otro tipo de tareas, por ejemplos F8 para pasar a la siguiente pantalla del listado de operaciones.

Por ejemplo, para accede a una utilidad muy usada como es el ISPF/PDF, para poder trabajar con data sets, nos bastaría con teclear P y pulsar ENTER.

Esto hará que nos aparezca la pantalla de la Figura 1.8 que nos muestra las diferentes opciones del ISPF/PDF.

Figura 1.8 Opciones de ISPF

Ahora, la forma de seleccionar la opción deseada es por medio de números en vez de letras como antes pero funciona de la misma manera.

Estando en la pantalla de la Figura 1.7, si sabemos que la opción a la que queremos acudir es directamente la 0 del ISPF/PDF, en vez de tener que teclear P, pulsar ENTER, teclear 0 y pulsar ENTER, podemos directamente teclear P.0 y pulsar ENTER. De esta forma podemos ir rápidamente a la opción que queramos siempre que sepamos el camino que hay que seguir. Por ejemplo es muy usada la opción P.3.4 para buscar data sets.

También desde la pantalla principal del ISPF/PDF, Figura 1.8, podemos ir directamente a una opción, por ejemplo tecleando 3.4. La P en este caso no hay que introducirla ya que sirve para ir a la pantalla principal del ISPF que es donde nos encontramos.

Otra opción muy usada es la SD que nos permite acceder a la utilidad llamada “System Display and Search Facility” o SDSF. Esta utilidad permite listar y buscar diferentes jobs que se están ejecutando, salidas de jobs ya ejecutados, estado de *Started Tasks*, TSU conectados, etc.

En la Figura 1.9 podemos ver la pantalla inicial del SDSF que nos permite acceder por ejemplo a la cola de usuarios activos, tanto *Started Tasks* como TSUs, colas de entrada, salida, etc.

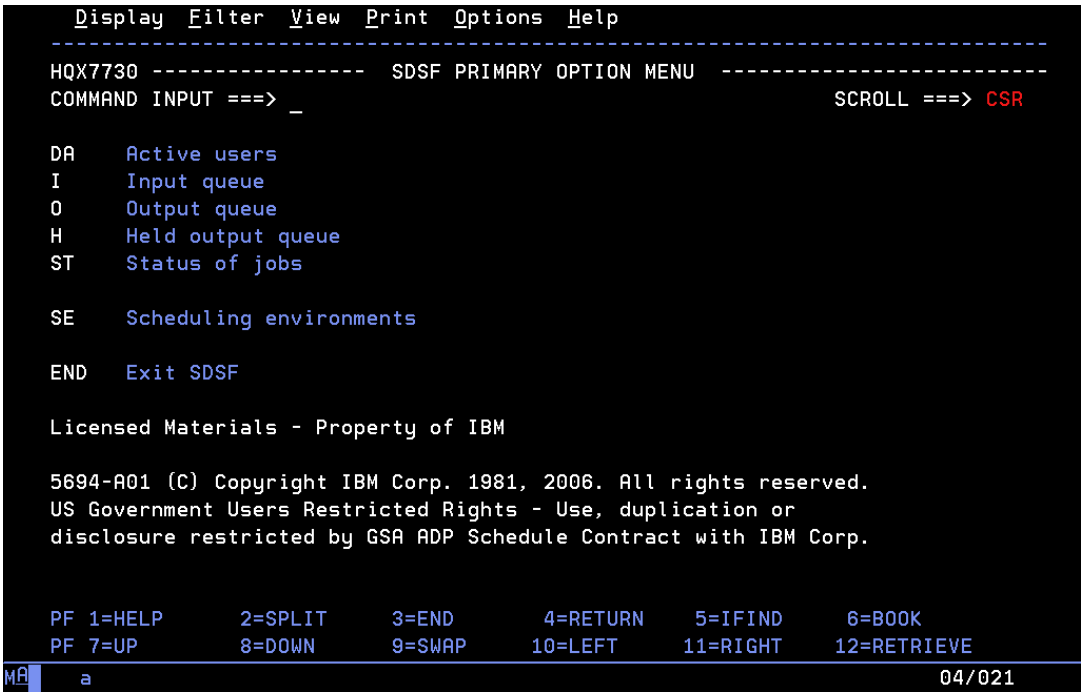


Figura 1.9 Pantalla inicial del SDSF

Por ejemplo, si nos dirigimos a la opción DA, podemos ver los usuarios activos como las *Started Tasks*, los usuarios conectados al TSO o TSUs y los jobs.

Por ejemplo, en la Figura 1.10 podemos ver una lista de *Started Tasks* como pueden ser diferentes CICS que están en ejecución, CTGs, etc.

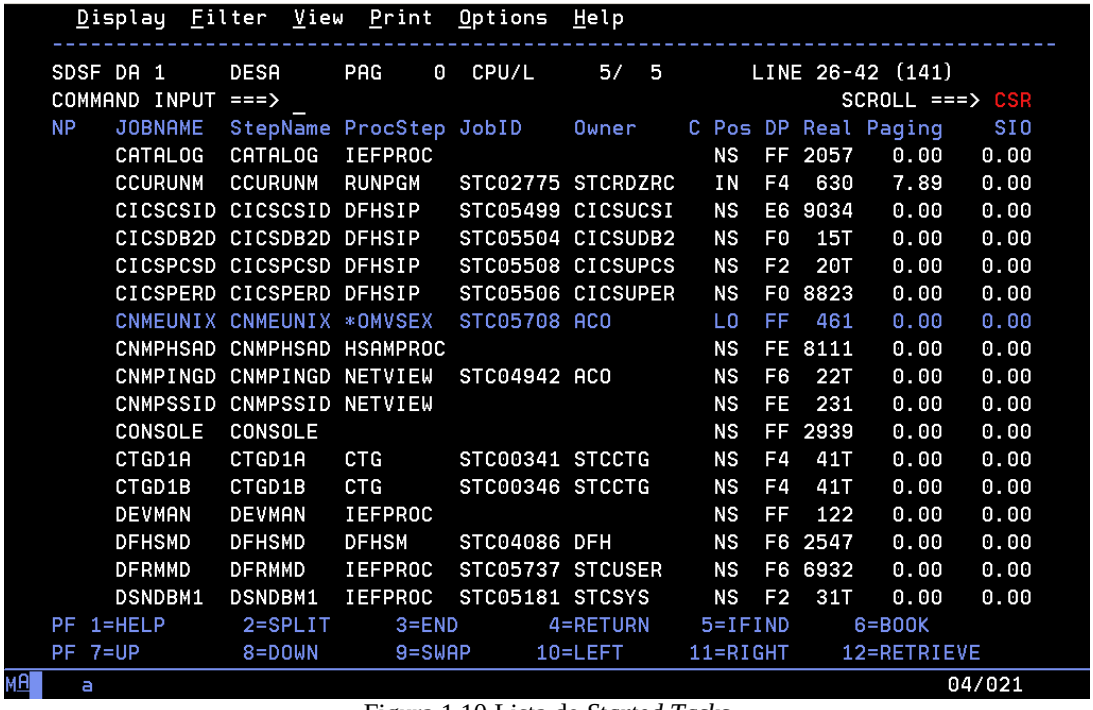


Figura 1.10 Lista de *Started Tasks*

Podríamos seleccionar una de estas *Started Tasks* y ver más detalles sobre ellas como posibles errores que se hayan producido en el CICS, conexiones realizadas a un CTG, etc.

Desde el SDSF es desde donde arrancaremos y pararemos los CTGs y listaremos sus estadísticas.

1.2.- Soluciones propuestas y solución elegida

Entre las diferentes soluciones que satisfacen parte o todos los objetivos que buscamos con nuestro proyecto, nos encontramos con dos grupos de soluciones diferentes.

Por un lado tenemos los conectores y adaptadores que permiten a sistemas externos realizar peticiones a CICS. Normalmente estos conectores o adaptadores permiten desde un servidor de aplicaciones como puede ser WebLogic o Websphere conectarse a un mainframe IBM y hacer peticiones al CICS. Esto permite, aparte de integrar la lógica CICS con otros sistemas externos, mejorar la presentación a la hora de hacer peticiones a transacciones CICS y recibir los datos que este nos devuelva. Por ejemplo, podemos arrancar una transacción pulsando un botón o link sin tener que acordarnos de su identificador. Los datos que nos devuelva los podemos mostrar en una tabla por la que nos podemos desplazar usando la rueda del ratón en vez de las teclas PF.

En la mayoría de los casos, estas soluciones necesitan tener instalado CICS Transaction Gateway en nuestro sistema.

Entre las diferentes soluciones encontradas de este tipo tenemos las siguientes:

- Librados – JCA Plus Adapter for CICS
  - Adaptador desarrollado por Oracle
  - Permite, en un principio desde cualquier servidor de aplicaciones conectarse a un mainframe y hacer peticiones CICS
  - Necesita que esté instalado el CICS Transaction Gateway
- Atunnity CICS Adapter
  - Solución desarrollada por la empresa Attunity
  - Permite, por medio de peticiones XML, que pueden realizarse desde .NET, Java, etc., realizar peticiones CICS a un servidor Attunity instalado en el mainframe
- JBI4CICS
  - Solución desarrollada por la empresa Imola Project Partners
  - Conector que permite, sobre un Enterprise Service Bus, Bus de Servicios de Empresa, llamado OpenESB, realizar peticiones CICS
  - Necesita del CICS Transaction Gateway para funcionar

Las desventajas que plantean estas soluciones son varias:

- Necesitamos tener instalado algún otro elemento como un servidor de aplicaciones o un bus ya que no podemos utilizarlo desde aplicaciones de escritorio.
- Ante cambios en la versión de CICS, nos podemos quedar sin servicio temporal o definitivamente hasta que se adapten a los cambios producidos.

En la línea de este tipo de soluciones, y viendo que muchas de ellas lo necesitan, vamos a ver el CICS Transaction Gateway.

El CICS Transaction Gateway, o CTG, es un componente de la empresa IBM que se instala en el mainframe y actúa como intermediario entre el CICS y sistemas externos para enviar y recibir peticiones.

Aparte del componente instalado en el mainframe, también suministra un conector que puede ser desplegado en un servidor de aplicaciones y una sería de librerías que pueden ser utilizadas desde aplicaciones de escritorio o por medio de Servlets, de forma que nos provee de una API para realizar peticiones al CICS.

Las ventajas que ofrece está solución son varias:

- Es de IBM con lo que aseguramos una compatibilidad completa con el resto del sistema.
- Por la misma razón, nos aseguramos el funcionamiento ante posibles cambios en la versión del CICS ya sea desde la versión que tengamos actualmente o actualizándonos a las nuevas versiones que puedan aparecer.
- Suministra tanto conectores como librerías para poder realizar peticiones tanto desde servidores de aplicaciones como desde aplicaciones de escritorio.
- Permite enviar y recibir peticiones desde aplicaciones escritas en los siguientes lenguajes:
    - .NET
    - C
    - C++
    - COBOL
    - Java

El mayor inconveniente del CICS Transaction Gateway es que, debido a la arquitectura de nuestro sistema, como veremos más adelante, no es posible realizar peticiones de transacciones conversacionales o pseudo-conversacionales, es decir, aquellas que presentan pantallas. Este problema también lo vamos a tener en la mayoría de las otras soluciones vistas al hacer uso estas del CICS Transaction Gateway.

El CICS Transaction Gateway no viene junto con la licencia de CICS TS por lo que hay que pagar por él.

Viendo las desventajas que ofrecen otras soluciones, además de que muchas de ellas necesitan el CICS Transaction Gateway, junto con las ventajas que esta última solución presenta, nos decantemos por el CICS Transaction Gateway entre este tipo de soluciones. El inconveniente de no poder llamar a transacciones con pantallas lo solucionaremos con la segunda solución elegida.

Para solucionar el inconveniente del CICS Transaction Gateway de no poder llamar a transacciones que presentan pantalla lado tenemos los maquilladores de pantallas 3270.

Estas soluciones reciben tráfico 3270 y lo convierten en páginas Web o en pantallas “normales” de un ordenador de forma que podamos usar el ratón, utilizar atajos de teclado para copiar, pegar, etc. Con esto logramos “modernizar” la interacción con la lógica CICS ya que transformamos una interfaz textual en una interfaz gráfica.

Entre este tipo de soluciones tenemos las siguientes:

- Micro Focus OnWeb
    - Producto de la empresa Micro Focus
    - Muestra las “pantallas verdes” de un mainframe como páginas web o interfaces .NET
- Passport Web-to-Host
    - Solución de la empresa Passport
    - Permite conectarnos al mainframe desde un navegador Web
- OpenText Terminal Emulation
    - Producto de la empresa OpenText
    - Permite accede al mainframe desde el escritorio o desde un navegador web

El mayor problema de este tipo de soluciones es que normalmente introducen un retardo significativo en la comunicación entre el cliente web y el mainframe, que puede ir desde varios cientos de milisegundos hasta algunos segundos lo que hace que la comunicación no sea fluida y que la productividad caiga drásticamente.

Tampoco podemos hacer uso directo de los datos recibidos en otros sistemas externos sino que desde una aplicación Java, por ejemplo, abría que leer el código HTML o XML que nos llega y extraer los datos para poder operar con ellos.

Una solución que no sigue exactamente la línea de las anteriores pero que nos va a permitir llamar a transacciones con pantallas por medio del CICS Transaction Gateway es el CICS Service Flow Feature.

Este producto de IBM nos permite agregar múltiples llamadas a transacciones CICS en un solo flujo. También permite automatizar ciertas interacciones con pantallas 3270, como introducción de las opciones necesarias para llegar a la operación que queremos realizar, de forma que podamos reproducir este flujo simplemente pasándole los datos de entrada necesarios.

De esta forma podemos grabar flujos que simulen la interacción con una transacción con pantallas, instalar este flujo en el CICS TS y luego invocarlo por medio del CICS Transaction Gateway.

Esto nos permite llamar a transacciones conversacionales o pseudo-conversacionales, cuya interacción con el usuario hemos grabado previamente, desde el CICS Transaction Gateway. También mejora la eficiencia al juntar múltiples peticiones en una sola y automatizar ciertas interacciones con la transacción.

Nos permite además invocar estos flujos desde servicios Web. Aunque no es una de las características del SFF que más vamos a utilizar en este proyecto, también explicaremos como utilizarla.

Pongamos por ejemplo que en una transacción bancaria tenemos que introducir una opción para ir a las cuentas de usuarios, meter el número de cuenta sobre la que queremos operar, introducir la opción de agregar saldo y por último introducir el saldo que queremos añadir a la cuenta.

Grabando estas interacciones con la transacción bancaria en un solo flujo simplemente le pasaríamos como datos de entrada el número de cuenta y el saldo que queremos añadir. Él se encargaría de introducir de forma automática las opciones para ir a las cuentas de usuario y para acceder a la operación de agregar saldo que hemos grabado, mejorando así la eficiencia.

En la Figura 1.11 vemos como sería un flujo “normal” donde tenemos que interactuar varias veces con el CICS para obtener el resultado requerido.

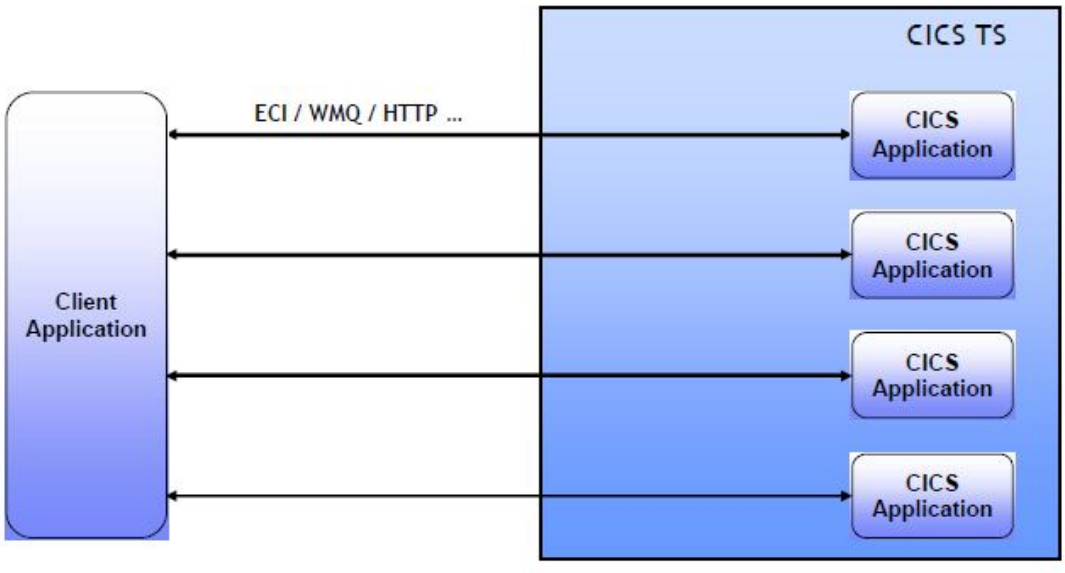


Figura 1.11 Múltiples interacciones y peticiones para realizar una operación

En la Figura 1.12 vemos que, grabando y reproduciendo el flujo anterior por medio del CICS Service Flow Feature, sólo necesitamos una interacción con el CICS, para decirle que flujo queremos ejecutar y pasarle los datos de entrada, en caso de necesitarlos, y este nos devuelve los resultados obtenidos.

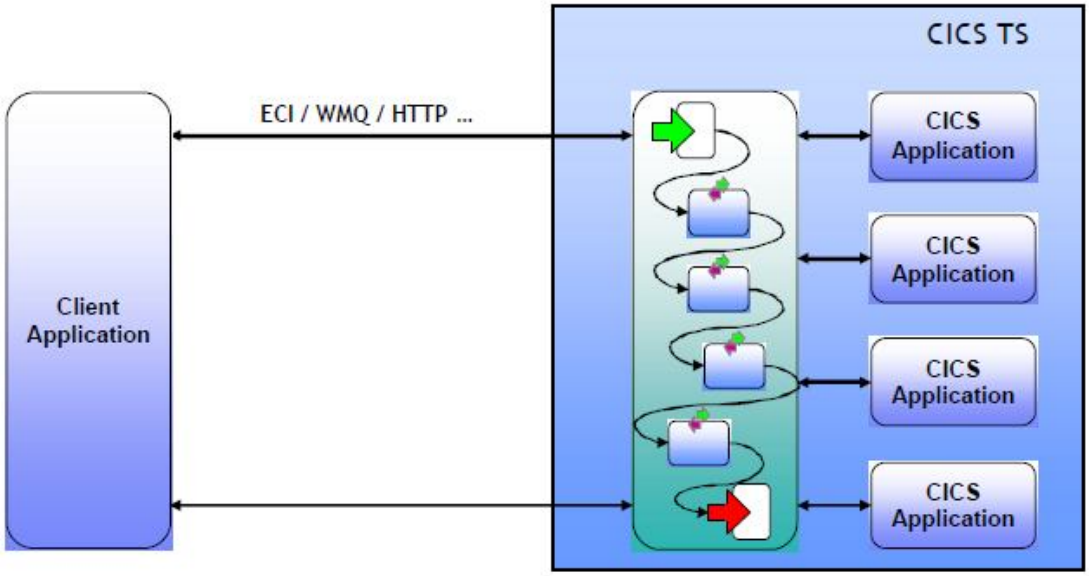


Figura 1.12 Una sola interacción y petición para realizar una operación

Al llamar a los flujos por medio del CICS Transaction Gateway, vamos a poder integrar esta solución son otros sistemas externos, ya sean Java, .NET, etc. de forma que podemos operar directamente con los datos que recibimos.

Al igual que con el CICS Transaction Gateway, al ser un producto de IBM la compatibilidad con cada versión de CICS que va apareciendo es completa.

El único inconveniente es que cada uno de los flujos que más adelante queremos utilizar hay que grabarlo previamente. Aunque esto al principio puede parecer costoso, grabar un flujo con 5 o 6 pantallas no cuesta más de 15 minutos, tiempo que a la larga vamos a recuperar al no tener que volver a introducir los mismos datos y opciones de navegación entre pantallas una y otra vez.

Hay que indicar que el CICS Flow Feature consta de dos partes.

Por un lado tenemos el Service Flow Runtime, o SFR, que es un componente que se instala en el mainframe y que permite ejecutar los flujos grabados. Este componente es gratuito y lo podemos instalar en cualquier mainframe en el que tengamos una licencia del CICS TS.

Por otro lado tenemos el Service Flow Modeler, o SFM, que es la herramienta que utilizamos para grabar los flujos y que se utiliza desde un ordenador personal. El SFM forma parte del módulo Enterprise Service Tools que a su vez forma parte del Rational Developer for System z. Para poder ejecutar este último necesitamos el Rational Business Developer. Estos programas son de pago y permiten realizar otras muchas tareas como desarrollar programas COBOL para CICS, crear servicios web que podemos ejecutar en el mainframe, etc.

Por estas razones elegimos el CICS Service Flow Feature para solucionar la limitación del CICS Transaction Gateway de no poder trabajar con transacciones que presentan pantallas.

1.3.- Hardware y software necesario

Para realizar este proyecto vamos a necesitar dos máquinas.

Por un lado vamos a necesitar un mainframe donde instalar el CICS Transaction Gateway y el Service Flow Runtime. Por otro lado vamos a necesitar un ordenador personal que vamos a utilizar para desarrollar y probar aplicaciones que se conecten al mainframe.

Opcionalmente podríamos hacer uso de otra máquina donde tener instalado un servidor de aplicaciones desde donde hacer las pruebas aunque los propios programas que vamos a utilizar para desarrollar aplicaciones nos permiten simular un servidor de aplicaciones desde donde probarlas.

Vamos a especificar los componentes hardware que vamos a necesitar en el mainframe así como en el ordenador personal que utilicemos para conectarnos al mainframe y realizar las pruebas y los componentes software que vamos a necesitar tanto en el mainframe como en el ordenador personal.

1.3.1.- Hardware

1.3.1.1.- Hardware en la parte mainframe

El único hardware que vamos a necesitar en la parte mainframe es el propio mainframe que va a ser un IBM System z10 Business Modelo M03 similar al que podemos ver en la imagen.

Figura 1.13 Mainframe System z10

En concreto, el nuestro dispone de las siguientes características:

- 16 GB de memoria RAM
- 1 procesador IFL
  - Procesador especial para poder instalar Linux para System z
  - No confundir con la parte Unix del z/OS
- 1 procesador zIIP
  - Procesador especial para DB2

1.3.1.1.- Hardware en la parte del ordenador personal

El hardware necesario en el ordenador personal es el propio ordenador que cumpla los requerimientos necesarios para poder ejecutar el software que veremos a continuación.

En particular, vamos a utilizar un ordenador con las siguientes características:

- Pentium 4 a 3.20 GHz
- 80 GB de disco duro
- 1.5 GB de RAM
- Lector de DVDs

Este ordenador cumple todos los requerimientos de hardware del software que vamos a utilizar. Aunque el Rational Business Developer pide una cantidad recomendad de RAM de 2 GB, la cantidad necesaria es de 1 GB y la experiencia nos dice que 1.5 GB es suficiente para su funcionamiento.

1.3.2.- Software

1.3.2.1.- Software en la parte mainframe

El software que vamos a necesitar en la parte mainframe es el siguiente, las versiones podrían ser otras siempre y cuando sean compatibles entre ellas:

- z/OS v1.8
- CICS Transaction Server v3.2
  - Esta es la versión mínima para poder usar conexiones IPIC, que veremos más adelante
- CICS Service Flow Runtime para CICS TS v3.2
- CICS Transaction Gateway v7.2
  - Para poder hacer uso de conexiones IPIC necesitamos como mínimo la versión 7.1
- DB2 v8.1
  - Aunque no es estrictamente necesario para nuestro proyecto, si que va a ser utilizado por alguno de los programas CICS que utilicemos.

1.3.2.2.- Software en la parte del ordenador personal

En el ordenador personal vamos a necesitar el siguiente software:

- Personal Communications v5.8
  - Emulador de terminal 3270
- Rational Business Developer v7.5.1.5
  - Necesario para poder instalar el Rational Developer for System z
- Rational Developer for System z v7.6.0
  - Con el modulo Enterprise Service Tools
- Microsoft Visual Studio 2005 v8.0.50727.42
  - Suite de desarrollo de aplicaciones .NET
- NetBeans IDE v6.5

    - Suite de desarrollo de aplicaciones Java
- BlueZone Secure FTP v5.0
    - Permite transferir ficheros o data sets entre un ordenador personal y un mainframe

El emulador utilizado tiene varias características que vale la pena comentar:

- El ENTER habitual, para validar o realizar una acción, no se simula con la tecla habitual sino con la tecla Control, o Intro del teclado numérico
- La tecla Intro habitual nos lleva al primer campo de la siguiente línea donde podemos introducir texto
- Con la tecla Fin en un campo de entrada, borramos todo lo que hay desde el punto donde tenemos el cursos hasta el final del campo
- En una sesión CICS, con la tecla Pausa borramos la pantalla

En la Figura 1.14 podemos ver el emulador Personnal Comunnications v5.8.

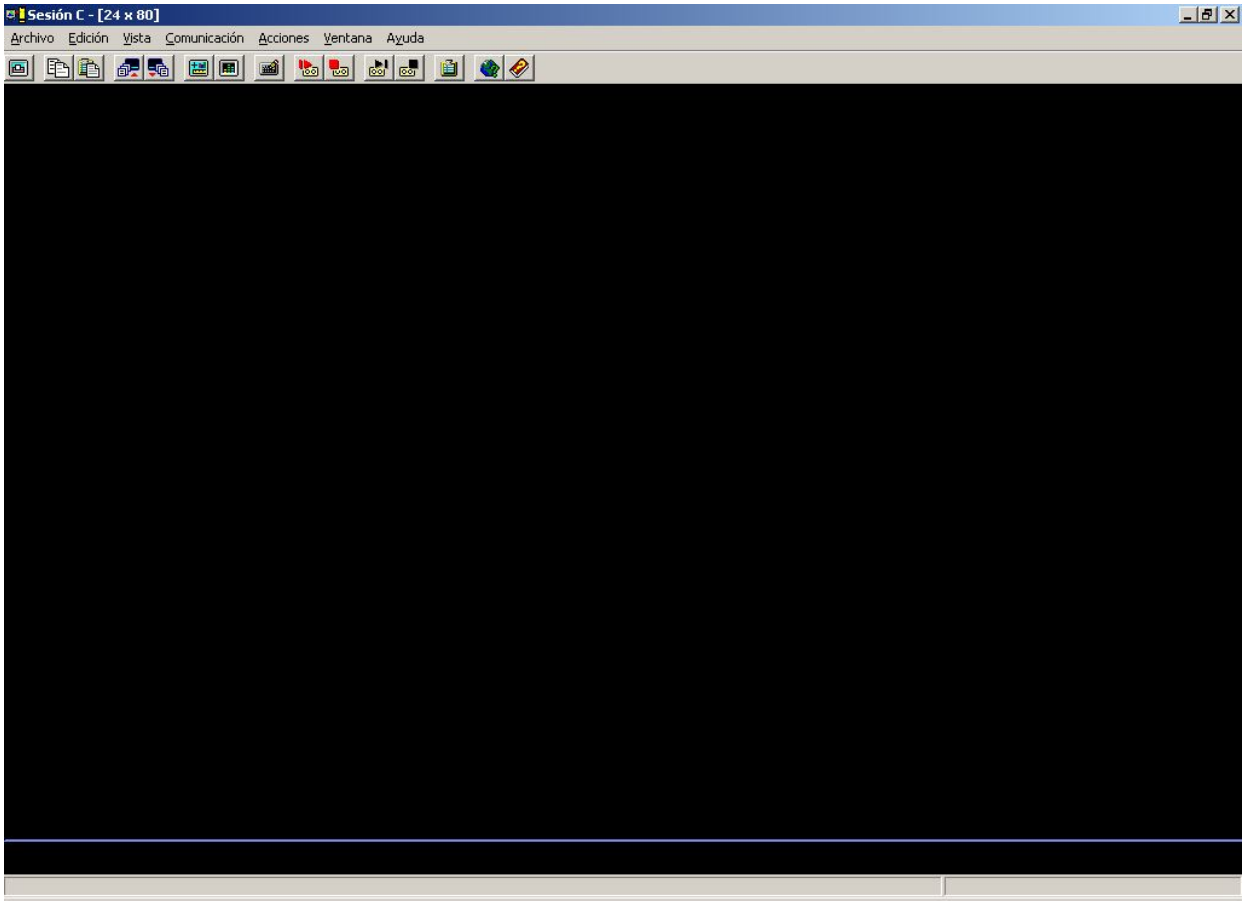


Figura 1.14 Personnal Comunnications v5.8

2.- CICS Transaction Gateway

2.1.- Introducción a CICS Transaction Gateway

El CICS Transaction Gateway, también conocido como CICS TG, o CTG que será como lo llamemos a partir de ahora, es un componente de la empresa IBM que se instala en el mainframe y actúa como intermediario entre el CICS y sistemas externos para invocar servicios en el primero.

Esto permite integrar la lógica de negocio de CICS con otros sistemas externos al mainframe, como pueden ser aplicaciones escritas en diferentes lenguajes de programación, para así mejorar y modernizar el sistema de información que se desee.

El CTG consta de un componente que actúa como servidor escuchando peticiones que lleguen de los clientes. Este componente invoca los servicios correspondientes y devuelve los resultados obtenidos al cliente que le ha hecho la petición.

Por otro lado consta de una parte cliente que son un grupo de conectores, llamados *resource adapter* según la especificación JCA, utilizada para conectar servidores de aplicaciones Java con sistemas de información de empresa, y librerías que van a permitir utilizar las API's necesarias para realizar peticiones al CTG. Este grupo de conectores y librerías va a permitir realizar peticiones al CTG desde aplicaciones escritas en diferentes lenguajes como Java, .NET, etc. ya sean desarrolladas como aplicaciones de escritorio o como aplicaciones Web desplegadas en un servidor de aplicaciones.

En la Figura 2.1 podemos ver los diferentes componentes que se van a emplear para realizar una petición al CICS por medio del CTG.

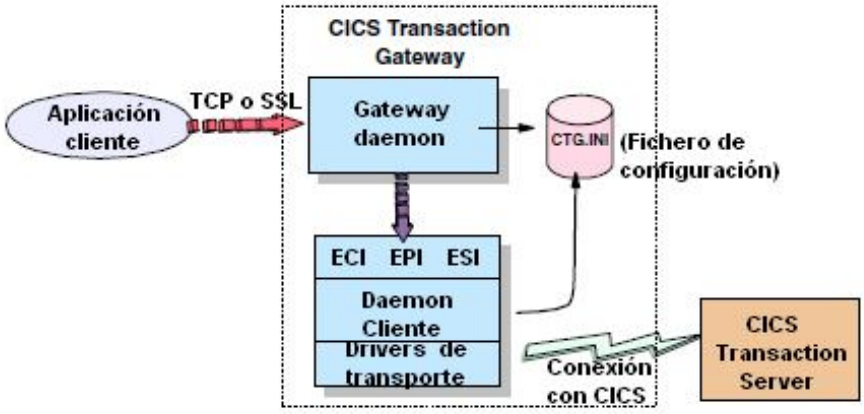


Figura 2.1 Componentes que se utilizan en una petición con CTG

Primero tenemos la aplicación que va a realizar la petición. Esta aplicación podrá ser de escritorio o tipo Web y estar escrita en variados lenguajes de programación como veremos más adelante.

La petición viaja por TCP o TCP con SSL hasta un demonio de CTG que está escuchando peticiones.

Este demonio y el demonio utilizado para conectarse al CICS se configuran por medio de un fichero .ini.

El CTG obtiene los datos necesarios de la petición dependiendo de la API utilizada. Esta API puede ser ECI, EPI o ESI como veremos posteriormente. Comprueba que todo está correcto y, en caso de ser así, realiza una petición al CICS para lo que inicia un demonio cliente.

El tipo de conexión entre el CTG y el CICS puede variar bastante dependiendo de la arquitectura que tengamos montada en nuestro sistema, como veremos más adelante.

La petición llega al CICS el cual la procesa y los resultados obtenidos viajan en sentido inverso hasta el cliente.

CTG tiene numerosas características de las que podemos hacer uso siendo algunas de las más importantes las siguientes:

- Conexión por TCP o SSL entre el cliente y el CTG
- Peticiones a programas basadas en COMMAREA o en Canales y Contenedores
- Peticiones a transacciones conversacionales o pseudo-conversacionales
- Soporte para lenguajes o interfaces de programación:
  - Java
  - .Net
  - C
  - C++
  - COBOL
  - COM
- Autentificación por usuario o usuario y contraseña a la hora de realizar peticiones
- Soporte para transacciones XA y two-phase commit, o commit de dos fases

En general vamos a hacer uso de todas ellas excepto de la posibilidad de realizar peticiones a transacciones conversacionales o pseudo-conversacionales y el soporte de transacciones XA y two-phase commit.

La primera de ellas no la vamos a utilizar debido a que, como veremos más adelante, la arquitectura de nuestro sistema no lo permite.

La segunda de ellas no la vamos a utilizar al no tener nuestro sistema preparado para utilizar este tipo de transacciones y de protocolo.

Tampoco vamos a hacer uso de todos los lenguajes de programación que permiten acceso al CTG sino que vamos a utilizar Java y .Net únicamente porque consideremos que son las opciones más utilizadas. Además, el uso de la API de .Net para realizar peticiones al CTG y el del resto de lenguajes, exceptuando Java, es el mismo.

Nosotros vamos a trabajar con la versión 7.2 del CTG aunque la versión más actual en estos momentos es la posterior a esta que es la 8.0. En ambas versiones existe la opción de instalar el CTG en un sistema operativo z/OS, el llamado CTG para z/OS, o en un sistema operativo Windows, Unix o Linux, el llamado CTG multiplataforma. La licencia de CTG que poseemos sólo nos permite instalar el CTG en un sistema operativo z/OS.

En general los cambios introducidos con esta nueva versión 8.0 no son demasiado importantes exceptuando el soporte en C, C++ y .Net de los Canales y Contenedores que veremos más adelante.

2.2.- Posibilidades en las arquitecturas y API's

2.2.1.- Posibilidades en las arquitecturas

El CTG nos provee de dos posibles arquitecturas a la hora de instalarlo. El Modo Local y el Modo Remoto. Ambos Modos nos van a permitir instalar el CTG de z/OS o el CTG multiplataforma, lo que va a posibilitar el uso de unas API's u otras.

2.2.1.1.- Modo Local

Este Modo se da cuando tanto el CTG como el cliente que realiza las peticiones, ya sea una aplicación de escritorio o un servidor de aplicaciones, se encuentran instalados en la misma máquina.

Este modo nos permite la posibilidad de instalar el CTG tanto en un servidor con Windows, Unix o Linux como en un mainframe System z ya sea sobre z/OS o sobre la versión Linux para System z. Para poder instalar el CTG en Modo Local sobre z/OS, el servidor de aplicaciones que utilicemos tiene que tener una versión para este sistema operativo cosa que actualmente sólo cumple el WebSphere Application Server, propiedad de IBM.

En la Figura 2.2 podemos ver el CTG y un servidor de aplicaciones en Modo Local instalado sobre una plataforma distribuida, es decir, sobre Windows, Unix o Linux.

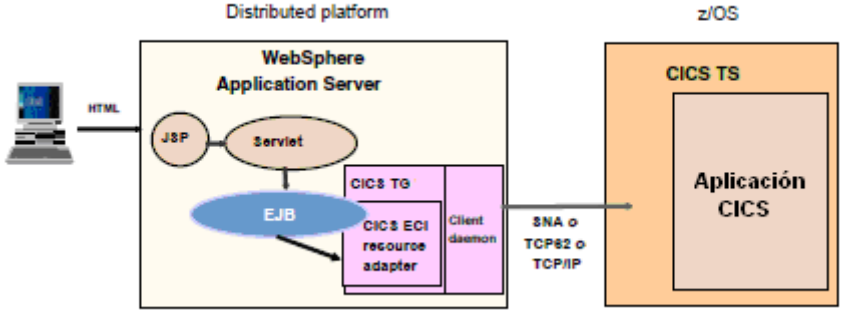


Figura 2.2 CTG multiplataforma en Modo Local

Vemos como en este caso los modos de conectar el CTG al CICS son por medio de SNA, modo de conexión propietario de IBM, TCP62, que es SNA sobre TCP/IP, o TCP/IP.

En la Figura 2.3 podemos ver también el Modo Local pero esta vez con el CTG y el servidor de aplicaciones instalados sobre z/OS.

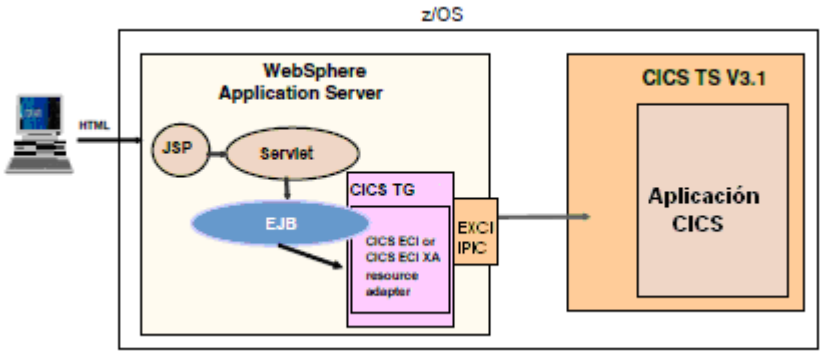


Figura 2.3 CTG sobre z/OS en Modo Local

Esta opción nos permite conectar el CTG y el CICS por medio de EXCI o IPIC.

Por último, en la Figura 2.4 podemos ver el CTG en Modo Local con el servidor de aplicaciones, ambos sobre un sistema operativo Linux instalado en el mainframe.

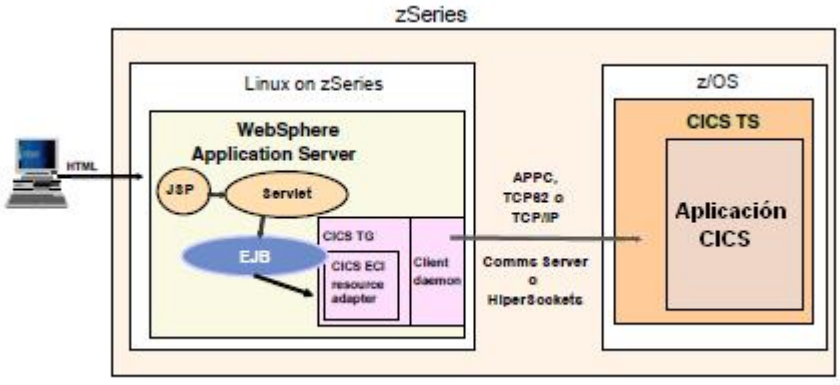


Figura 2.4 CTG en Modo Local sobre Linux para mainframes

En esta opción vemos como las posibilidades de conexión entre el CTG y el CICS aumentan permitiéndonos conectarnos por APPC, TCP62, TCP/IP, Communications Server o HiperSockets, siendo APPC, TCP62 e HiperSockets propietarias de IBM.

2.2.1.2.- Modo remoto

El Modo Remoto se da cuando la aplicación cliente, ya sea de escritorio o Web por medio de un servidor de aplicaciones, se encuentran en distinta máquina.

Al igual que antes, el CTG puede estar instalado sobre Windows, Unix o Linux, en este último caso en un mainframe o no, o en un mainframe sobre el sistema operativo z/OS aunque lo normal es que esté instalado en un mainframe sobre z/OS.

En la Figura 2.5 podemos ver un CTG sobre z/OS en Modo Remoto. Debido a que sólo tenemos licencia de CTG para z/OS y no tenemos licencia de WebSphere Application Server para z/OS, esta es la única arquitectura que vamos a poder montar.

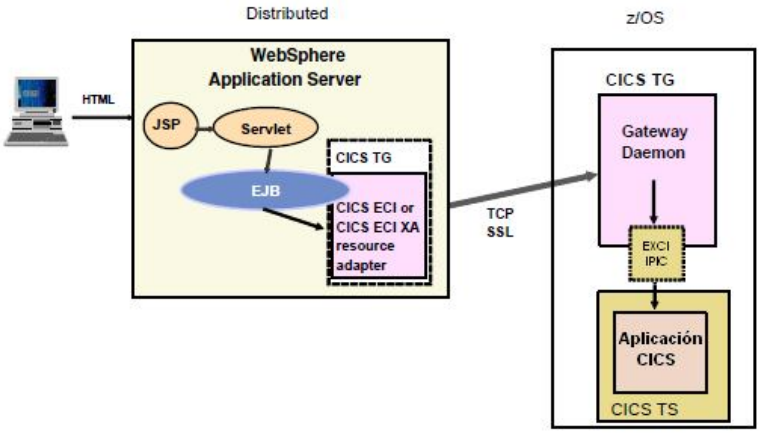


Figura 2.5 CTG en Modo Remoto sobre z/OS

Como podemos observar, en esta arquitectura tenemos la posibilidad de conectar la aplicación cliente y el CTG por medio de TCP o SSL. Las posibilidades de conexión entre el CTG y el CICS son por medio de EXCI o IPIC. Más adelante veremos las posibilidades de cada una.

En la Figura 2.6 podemos ver los diferentes tipos de topologías que podemos montar atendiendo a las conexiones que necesitamos.

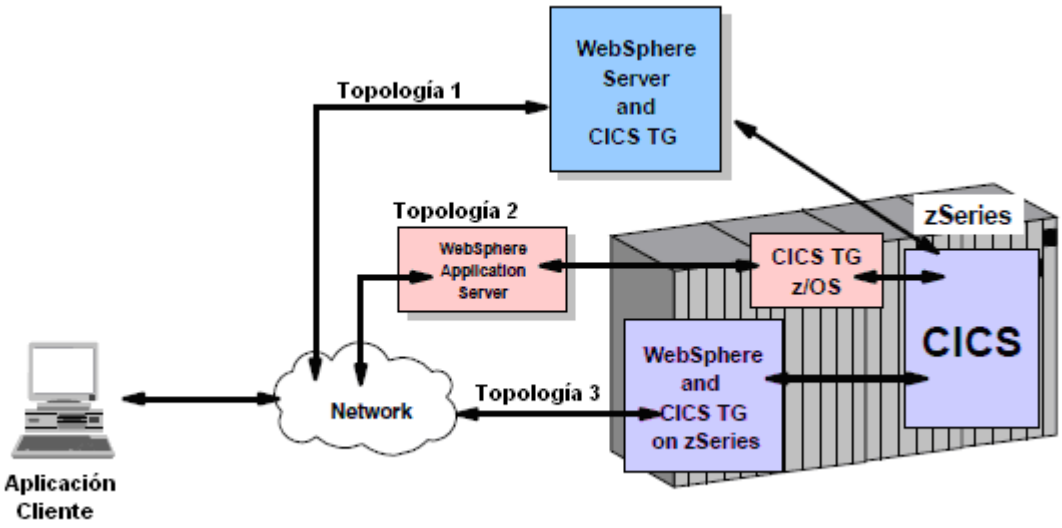


Figura 2.6 Topologías según las conexiones utilizadas

En la Topología 1 vemos como necesitamos una conexión entre el navegador Web y el servidor de aplicaciones. No sería necesaria ninguna conexión entre el servidor de aplicaciones y el CTG al encontrarse en la misma máquina, independientemente del sistema operativo que utilice esta máquina. También necesitamos una conexión entre el CTG y el CICS.

La Topología 2 necesita una conexión entre el servidor de aplicaciones y el CTG, o entre la aplicación de escritorio y el CTG, y otra entre el CTG y el CICS, a pesar de encontrarse en la misma máquina. Esta será la topología que utilicemos y como vemos es la única donde el servidor de aplicaciones y el CTG están en Modo Remoto.

Por último, la Topología 3 necesita, al igual que la 1, una conexión entre el navegador Web y el servidor de aplicaciones y otra entre el CTG y el CICS, independientemente de que el sistema operativo donde estén el servidor de aplicaciones y el CTG sea Linuz para System z o z/OS.

2.2.2.- Posibilidades en las API's

El CTG nos ofrece tres diferentes API's de las que podemos hacer uso para realizar peticiones de servicios CICS desde aplicaciones de escritorio o aplicaciones Web sin hacer uso de la tecnología JCA. Cada una de ellas nos va a permitir realizar un tipo de petición y, dependiendo del tipo de arquitectura y del sistema operativo donde tengamos instalado el CTG, podremos hacer uso de unas u otras API's. Estas API's se llaman ECI, EPI y ESI.

En caso de querer usar la tecnología JCA propuesta por Java para conectar el CTG con un sistema externo, utilizaremos la interfaz CCI, Common Client Interface o Interfaz de Cliente Comun, para realizar las peticiones al CTG. En caso de usar esta tecnología también necesitaremos usar los adaptadores CICS ECI o CICS ECI XA, dependiendo de si queremos hacer uso de transacciones XA o no.

Para hacer peticiones al CTG desde una aplicación Web, tenemos la opción de importar la librería correspondiente en cada aplicación que despleguemos o utilizar la tecnología JCA para hacer uso de un adaptador previamente desplegado en el servidor de aplicaciones. Por motivos que veremos más adelante, vamos a hacer uso de la primera opción.

2.2.2.1.- ECI

La API ECI, External Call Interface o Interfaz de Llamada Externa, nos va a permitir realizar peticiones a transacciones CICS no conversacionales, es decir, aquellas que no utilizan pantallas para la comunicación entre el CICS y el usuario. También nos va a permitir realizar peticiones para hacer *commit* o *rollback* sobre una LUW.

Los programas CICS invocados pueden utilizar tanto COMMAREA como Canales Contenedores para enviar y recibir datos.

La aplicación de usuario que haga uso de esta interfaz va a poder conectarse a varios servidores CICS al mismo tiempo y realizar varias peticiones de servicios CICS para que se ejecuten concurrentemente.

Los programas CICS invocados por medio de esta API van a hacer uso de DPL, o Distributed Program LINK, y deben seguir sus reglas. En un principio estas reglas no son muy restrictivas ya que son reglas como tener que pasarle la longitud de la COMMAREA, en caso de utilizarla, el nombre del programa CICS que queremos que se ejecute, etc. Siguiendo las indicaciones que más adelante veremos para realizar peticiones con esta interfaz no debemos preocuparnos de estas reglas aunque se pueden consultar en el documento SC33-1687 - *CICS Application Programming Guide*.

Para poder hacer uso de esta interfaz, en nuestras aplicaciones haremos uso de la librería ctgclient.jar, en caso de ser una aplicación Java, o las librerías ctgclient.dll y IBM.CTG.Client.dll en caso de ser una aplicación .NET, que son las dos opciones que vamos a utilizar.

2.2.2.2.- EPI

La API EPI, External Presentation Interface o Interfaz de Presentación Externa, permite a una aplicación de usuario instalar y borrar terminales virtuales 3270 en un servidor CICS. Esto nos

va a permitir realizar peticiones de transacciones conversacionales o pseudo-conversacionales, es decir, aquellas que hacen uso de pantallas para comunicarse con el usuario.

Debido a que utilizamos el Modo Remoto entre el la aplicación de usuario y el CTG, y que este está instalados sobre z/OS, no nos va a ser posible utilizar esta API.

2.2.2.3.- ESI

La API ESI, External Security Interface o Interfaz de Seguridad Externa, permite utilizar ciertas funcionalidades de seguridad desde una aplicación de usuario por medio de CTG. Entre estas funcionalidades tenemos las siguientes:

- Verificar que una contraseña es válida para cierto identificador de usuario
- Cambiar la contraseña almacenada para un identificador de usuario
- Comprobar si el usuario está revocado, esto suele ocurrir por introducir una contraseña errónea un número determinado de veces, normalmente tres. Cuando esto pasa el usuario no puede entrar al sistema hasta que un administrador de seguridad no le vuelva a dar permiso
- Determinar si una contraseña a expirado
- Obtener información sobre el identificador de usuario que está realizando las peticiones
- Establecer unas credenciales de seguridad predeterminadas para comunicarse entre la aplicación cliente y el servidor CICS

Para hacer uso de esta API, las credenciales de seguridad tienen que estar gestionadas por un External Security Manager, ESM, válido para CICS como RACF. Este ESM tiene que ser accesible para el servidor CICS particular sobre el que estamos trabajando.

Al igual que con la API EPI, debido a que utilizamos el Modo Remoto entre el la aplicación de usuario y el CTG, y que este está instalados sobre z/OS, no nos va a ser posible utilizar esta API.

2.2.2.4.- CCI

La API CCI, Common Client Interface o Interfaz de Cliente Común, permite a servidores de aplicaciones Java conectarse y hacer uso de un sistema de información de una empresa como puede ser el CICS. CCI no es propiedad de IBM sino que es parte de la tecnología JCA desarrollada por Java. Para hacer uso de esta interfaz deberemos desplegar en el servidor de aplicaciones un adaptador que nos de acceso al sistema de información específico que queremos utilizar.

El CTG para z/OS nos provee de dos adaptadores como son CICS ECI y CICS ECI XA. Ambos nos permiten utilizar la API ECI que hemos visto anteriormente con la diferencia de que el segundo nos da la posibilidad de utilizar transacciones XA. También hay un adaptador para poder hacer uso de la API EPI pero no se suministra con el CTG de z/OS ya que no podemos hacer uso de é. No existe un adaptador para hacer uso de la API ESI ya que no es posible usar esta API por medio de JCA.

Las restricciones a la hora de usar estos adaptadores para hacer peticiones son las mismas que si utilizásemos las librerías importándolas en cada aplicación exceptuando las siguientes:

- No podemos hacer uso de LUW's que llamen a más de una transacción. Esto significa que el valor para el parámetro Extend_Mode debe de ser ECI_NO_EXTEND
- Todas las peticiones deben de ser síncronas. Esto significa que sólo los tipos ECI_SYN y ECI_SYNC_TPN son válidos para el tipo de llamada

Para poder utilizar estos adaptadores, antes de desplegarlos en el servidor de aplicaciones, vamos a tener que establecer ciertos parámetros como la URL del servidor CICS, su puerto de escucha, el usuario y contraseña que se va a utilizar para realizar las peticiones, etc. Esto tiene varios inconvenientes:

- En caso de tener varios servidores CICS tenemos que desplegar tantos adaptadores como servidores tengamos
- Cada aplicación tiene que indicar el adaptador que va a usar por lo que, si queremos que una aplicación acceda a más de un CICS, tendremos que desplegar la aplicación varias veces, cada una haciendo uso de un adaptador
- El usuario utilizado para realizar las peticiones es fijo, lo que nos impide llevar en caso necesario una auditoría del uso que cada usuario hace del CTG
- Si al contraseña cambia, debido a que ha expirado o a que se ha modificado por razones de seguridad, habría que volver a desplegar el adaptador modificando los parámetro necesarios

Estos inconvenientes superan en mi opinión al inconveniente de tener que importar la librería correspondiente en cada aplicación Web que desarrollemos. Por esto vamos a usar esta segunda opción en caso de querer desarrollar una aplicación Web.

2.2.2.5.- Posibles API's según la arquitectura

Dependiendo del Modo de arquitectura utilizado y del sistema operativo sobre el que esté instalado el CTG, vamos a poder hacer uso de unas API's o de otras.

En general, si tenemos el CTG para multiplataforma instalado en Modo Local podemos hacer uso de cualquier API desde cualquier lenguaje de programación. Si el CTG está en Modo Remoto sólo vamos a poder usar las API's ECI y EPI desde Java y sólo la ECI desde C y .Net.

Si el CTG está instalado sobre z/OS, como es nuestro caso, sólo vamos a poder usar la API ECI desde Java y .Net si está en Modo Remoto y sólo desde Java en modo Local.

En la Tabla 2.1 podemos ver un resumen de las API's que podemos usar en Modo Local según el Sistema Operativo sobre el que está instalado el CTG.

API	C	C++	COBOL	COM	.Net	Java	JCA
Windows							
ECI	X	X	X	X	X	X	X
EPI	X	X	X	X	X	X	X
ESI	X	X	X	X	X	X	
Unix y Linux							
ECI	X	X	X		X	X	X
EPI	X	X	X		X	X	X
ESI	X	X	X		X	X	
z/OS							
ECI						X	X
EPI							
ESI							

Tabla 2.1 API's utilizables en Modo Local

En la Tabla 2.2 podemos ver las API's que tenemos la ocasión de utilizar en Modo Remoto dependiendo del sistema operativo sobre el que tengamos instalado el CTG.

API	C	C++	COBOL	COM	.Net	Java	JCA
Windows							
ECI	X				X	X	X
EPI						X	X
ESI						X	
Unix y Linux							
ECI	X				X	X	X
EPI						X	X
ESI						X	
z/OS							
ECI	X				X	X	X
EPI							
ESI							

Tabla 2.2 API's utilizables en Modo Remoto

Vemos como si tenemos el CTG instalado sobre z/OS en Modo Remoto, como es nuestro caso, sólo podemos hacer uso de la API ECI desde Java y JCA, .Net y C. Probaremos la API desde los lenguajes Java y .Net ya que la forma de utilizarla desde los lenguajes JCA y C es similar respectivamente a Java y .Net.

En la Figura 2.7 podemos ver de forma más gráfica las diferentes API's que podemos usar con el CTG multiplataforma así como las posibles conexiones entre el servidor de aplicaciones, o aplicación de escritorio, y el CTG, como desde esta al CICS.

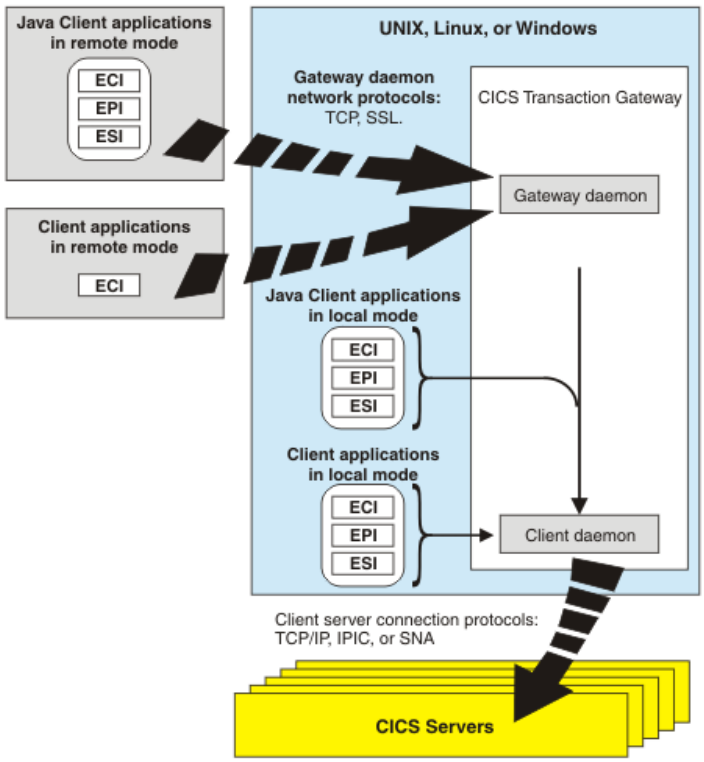


Figura 2.7 API's utilizables con el CTG multiplataforma

En la Figura 2.8 podemos ver las API's que se pueden usar por medio del CTG para z/OS y las conexiones posibles entre los diferentes componentes.

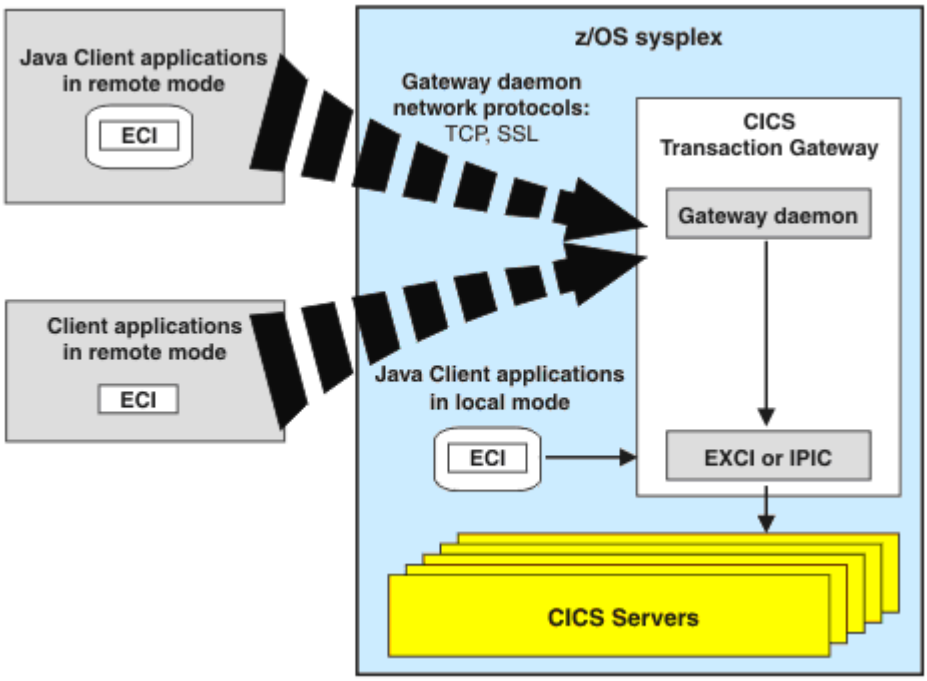


Figura 2.8 API's utilizables con el CTG para z/OS

Además de ver, como antes, que vamos a tener la posibilidad de utilizar la API ECI, también vemos que entre la aplicación cliente y el CTG vamos a poder usar TCP o SSL y entre el CTG y el CICS EXCI o IPIC.

2.2.3.- Tipos de conexiones

Como ya hemos visto, tenemos varias posibilidades de conexión, ya sea entre la aplicación de usuario y el CTG, como entre el CTG y el CICS.

La conexión entre la aplicación usuario y el CTG puede ser TCP o TCP con SSL. Esto no representa mayores diferencias en el uso y características del CTG que podemos utilizar. Simplemente añade un componente de autentificación y seguridad a la comunicación y los datos que se transmiten entre la aplicación cliente y el CTG.

Al estar tanto el CTG como el CICS instalados sobre z/OS, las posibles conexiones que podemos establecer entre ellos tienen que ser EXCI o IPIC. Las diferencias entre ambas conexiones a la hora de usar CTG son las siguientes:

- EXCI no soporta las peticiones de programas que usan Canales y Contenedores
- IPIC soporta transacciones XA sin tener que realizar ninguna configuración adicional
- Es más fácil habilitar seguridad en IPIC
- IPIC sólo soporta SSL cuando el CTG y el CICS están instalados en la misma máquina, como es nuestro caso

En general, la tendencia de IBM parece que es la de evolucionar hacia el uso de conexiones IPIC dejando de lado las conexiones EXCI en el uso del CTG.

Aunque veremos como configurar ambas conexiones, en general usaremos IPIC.

La mayor ventaja, desde mi punto de vista, de utilizar conexiones IPIC sobre EXCI es la posibilidad de utilizar una nueva interfaz para desarrollar programas CICS, como es la interfaz de Canales y Contenedores, sobre la antigua interfaz de COMMAREA. En el siguiente apartado pasamos a explicar las diferencias entre ambas y las ventajas de los Canales y Contenedores sobre la COMMAREA.

2.2.3.1.- Canales y Contenedores

En 1975 se introdujo la COMMAREA, comunnication area o área de comunicaciones, como un método para pasar datos de una aplicación a otra. Debido a que la opción de longitud de un comando CICS es generalmente expresado como un valor de longitud media palabra con signo, 15 bits en CICS., esto nos da un límite de longitud teórico de 32763 bytes. Esto significa que la máxima área de datos que se puede intercambiar es de 32KB. Esto no era un problema en sus inicios porque los datos de las aplicaciones estaban basados en 3270 y estos no consumían muchos recursos. El límite de 32KB era suficiente para manejar la propagación de datos entre aplicaciones.

Sin embargo, debido a que la integración entre aplicaciones CICS y elementos de las soluciones empresariales se están convirtiendo en la norma, las aplicaciones CICS cada vez necesitan procesar mayor cantidad de datos. La consecuencia de extender las aplicaciones CICS a nuevas soluciones empresariales es que las restricciones impuestas por el sistema en el tamaño de la COMMAREA pueden ser muy restrictivas.

La COMMAREA es un gran bloque de datos contiguo que contiene todos los datos pasados a un programa invocado, incluso si parte de los datos no son necesarios o son nulos.

Para el diseño moderno de aplicaciones la flexibilidad de las estructuras de datos es un componente básico que ofrece la posibilidad de acomodarse a futuros requerimientos empresariales.

Aunque hay diferentes opciones que solucionan el límite de 32KB de la COMMAREA, como pasar en la COMMAREA la dirección de un área de almacenamiento mayor, las diversas soluciones que se plantearon son válidas en algunos casos pero fallan o tienen problemas en otros por lo que se desarrolló una nueva interfaz para intercambiar datos entre aplicaciones CICS o desde el CICS hacia el exterior. Este nuevo enfoque es conocido como Channels y Containers o Canales y Contenedores. Fue introducido por primera vez en el CICS Transaction Server v3.2 y es posible utilizarlo a partir del CICS Transaction Gateway v7.1.

Un Canal es un nombre de referencia único a una colección de datos de aplicación contenidos en Contenedores. Un Canal es análogo a una COMMAREA pero sin tener las restricciones de esta última. Esta es una forma fácil de agrupar estructuras de datos que pueden ser pasados a una aplicación que invoques. CICS destruye un Canal cuando ya no puede ser referenciado más veces, esto es, cuando el Canal esté *out of scope*, fuera de ámbito, por lo que no tenemos que gestionar personalmente la memoria como con la COMMAREA.

Un Contenedor es un nombre de referencia a un área de almacenamiento manejada por CICS que puede contener cualquier forma de datos de aplicación. Un Contenedor puede ser teóricamente de cualquier tamaño, aunque en realidad está limitado por el direccionamiento de 31-bit de una región CICS, lo que implica un límite de 2GB por Contenedor, casi como si fuese ilimitado. Puede contener datos de cualquier tipo que las aplicaciones requieran. Una aplicación puede referenciar cualquier número de Contenedores.

Este nuevo enfoque nos permite intercambiar cantidades de datos superiores a 32KB al mismo tiempo que permite dar una estructura lógica a los mismos. También tiene una mejora sustancial con respecto a la COMMAREA y es que cuando enviamos un Canal sólo se envían los Contenedores nuevos o que se han modificado. No así con la COMMAREA donde se enviaba toda la COMMAREA cada vez. Esto hace que la cantidad de datos transferidos disminuya significativamente.

El único inconveniente es que necesitamos conocer el nombre de los Contenedores, no así del Canal normalmente, donde nos pasan los datos. De todas maneras, siempre es posible recorrer todos los Contenedores comprobando su contenido y quedándonos con la información que necesitemos.

La manera de manejar los Canales y Contenedores desde una aplicación Java, las aplicaciones .Net no pueden hacer uso de esta funcionalidad con la versión 7.2, se describe más adelante. En este proyecto no se va a tratar la manera de manejar Canales y Contenedores desde programas CICS aunque podemos encontrar como hacerlo en el manual EXEC CICS application programming interface.

2.3.- Instalación del CTG

2.3.1.- Requisitos de instalación

El CICS Transaction Gateway v7.2 no tiene grandes requisitos para su instalación. Simplemente necesitamos tener instalado un sistema operativo z/OS v1.8 o superior y un CICS Transaction Server v2.3 o superior. Aunque estas son las versiones mínimas para que el CTG v7.2 funcione, teniendo unas versiones superiores y ciertos APAR vamos a tener acceso a más funcionalidades. Un APAR es un Authorized Program Analisys Report, similar a un parche de software para corregir errores.

En la Tabla 2.3 podemos ver las diferentes funcionalidades a las que tenemos acceso dependiendo las versiones y APAR's que tengamos instalados.

Producto, versión y APAR's	**Funcionalidad**
z/OS v1.8 con APAR OA23163	APAR requerido para two-phase commit Transacciones XA
CICS TS v2.3 con APAR's PQ92943 y PK 17427	APAR's requeridos para aumentar límite de conexiones EXCI y two-phase commit Transacciones XA
CICS TS v3.1 con APAR PK17426	APAR requerido para two-phase commit Transacciones XA
CICS TS v3.2 con APAR's PK49015, PK49017, PK49021, PK49116, PK49490, PK51587, PK53783, PK55494, PK55495, PK55716, PK57726 y PK65134	Conexiones IPIC

Tabal 2.3 Funcionalidades dependiendo de las versiones y APAR's del software

Nuestro sistema se encuentra en el último caso por lo que nos permite usar tanto transacciones XA y two-phase commit como conexiones IPIC.

2.3.2.- Configuración del sistema

La instalación de CTG consta de dos partes.

La primera de ellas consiste en volcar los data sets y los directorios Unix necesarios para la instalación y funcionamiento del CTG desde la cinta donde IBM suministra sus productos hasta el mainframe. Esta parte ya estaba realizada al empezar a hacer el proyecto por lo que no ha sido posible documentarla. De todas maneras, los pasos necesarios para llevar a cabo este proceso vienen detallados en el siguiente documento que se puede descargar desde la página oficial de IBM http://www.ibm.com/: GI13-0512-02 - CICS Transaction Gateway for z/OS V7.2 Program Directory.

La segunda parte consiste crear los archivos de configuración necesarios para que funcione el CTG y personalizarlos con los valores adecuados para nuestra instalación. Estos pasos son los que vamos a describir a continuación.

2.3.2.1.- Reserva de puertos

Reservamos los puertos necesarios para nuestra instalación. Para ello se añadirán las siguientes líneas en el miembro PROFILE de la librería PRUEBAS.TCPIP.PARMLIB:

```
20061 TCP *              ; RESERVE CICS TO CTG1A
20062 TCP *              ; RESERVE CICS TO CTG1A
20066 TCP *              ; RESERVE CICS TO CTG1B
20067 TCP *              ; RESERVE CICS TO CTG1B
2006 TCP CTG1*  SHAREPORT  ; TCP/IP TO CTG
8050 TCP CTG1*           ; SSL TO CTG
```

Tal como están hechas las reservas, las únicas reservas válidas serían las dos últimas ya que les indicamos para quien lo reservamos. El resto servirían más bien para que si en otro momento queremos ver los puertos disponibles para instalar otro CTG u otro programa, saber que estos puertos los queremos utilizar en un futuro para estos CTGs.

Las primeras reservas las hacemos para conectar el CTG con el CICS. Las dos últimas, para reservar el puerto donde va a escuchar el CTG las peticiones que le lleguen del exterior.

En un principio, con reservar un solo puerto para conectar al CTG con el CICS y otro para que el CTG escuche peticiones sería suficiente. Hacemos más reservas en previsión de instalar más de un CTG, por lo menos dos. También hacemos reservas extra para conectar cada CTG a más de un CICS.

La penúltima línea indica que en el puerto 2006 vamos a tener dos o más CTGs escuchando en *Port Sharing*. Aunque esto lo explicaremos más adelante, lo que queremos hacer es tener dos o más CTGs escuchando en el mismo puerto de forma que ofrezcan alta disponibilidad. Esto quiere decir que se reparten la carga de trabajo o que si uno cae el resto siga escuchando peticiones en ese puerto. Para el usuario final esto es transparente.

La última línea la utilizamos para reservar el puerto 8050, por defecto, para escuchar peticiones que vengan por TCP sobre SSL.

Para que se lleven a cambio estos cambios en el sistema, habría que parar y arrancar todo el mainframe o reiniciar la LPAR donde queremos hacer los cambios.

2.3.2.2.- Configuración de la parte Unix

Una vez reservados los puertos que vamos a utilizar, el siguiente paso es crear y configurar con respecto a nuestra instalación un archivo .ini que va a servir para inicializar el CTG. Este archivo se almacena en la parte Unix del z/OS. Como personalmente me resulta más cómodo manejarme con una interfaz tipo línea de comandos de Unix que con los paneles de TSO/ISPF, accedemos a la parte Unix por medio de OMVS ejecutando en cualquier panel de TSO el comando tso omvs.

Como ya hemos dicho, los comandos para crear carpetas, copiar archivos, etc. son los mismos que en un sistema operativo Unix normal por lo que no los vamos a nombrar aquí.

A continuación vamos a ver los pasos que vamos a seguir para crear y modificar el archivo .ini como consideremos adecuado.

1.- Creamos el siguiente directorio donde vamos a almacenar los diferentes archivos .ini que podamos crear.

/etc/ctgvar/

2.- Copiamos el siguiente archivo

/usr/lpp/cicstg/v7r2m0/bin/ctgsamp.ini

a

/etc/ctgvar/

3.- Renombramos el siguiente archivo

/etc/ctgvar/ctgsamp.ini

a

/etc/ctgvar/ctg1a.ini

El número 1 se lo damos para distinguirlo de otros posibles CTGs, o grupos de CTGs en *Port Shared*, que podamos instalar más adelante.

La letra A se la damos para diferenciarlo dentro del grupo de CTGs que instalemos más adelante en *Port Shared*.

Nos tenemos que acordar de este nombre, y de la ruta donde está almacenado, ya que lo necesitaremos más adelante.

4.- Configuramos el archivo .ini según los parámetros de nuestra instalación.

Para poder leer y modificar un archivo utilizamos el comando oedit <nombre_archivo>.

Esto nos llevará a una pantalla similar a la que hemos visto anteriormente de ISPF/PDF pudiendo utilizar las mismas técnicas para copiar líneas, borrarlas, etc.

A continuación vamos a ver como quedaría el archivo una vez configurado e intentaremos explicar cada uno de los parámetros y valores que hemos establecido.

Hay que indicar que las líneas que empiezan por Ñ o, en una línea, el texto que va después de una Ñ, son comentarios. El Documento 2.1 muestra el fichero .ini con los parámetros adecuados para nuestra instalación.

```
Ñ Sección para indicar algunos parámetros que necesita el CTG
SECTION PRODUCT
      applid = CTG1A                Ñ Establece el APPLID del CTG
                                    Ñ Es el "nombre" del CTG
      applidqualifier = VTAM        Ñ El cualificador APPLID del Gateway
                                    Ñ Daemon
```

```
                                   Ñ Es el “nombre” de la red donde está
                                   Ñ conectado
                                   Ñ Ya definido en el sistema
      defaultserver = CICS1        Ñ Nombre del servidor CICS al que se va a
                                   Ñ conectar por defecto
                                   Ñ Ya definido en el sistema
ENDSECTION

Ñ Sección para indicar ciertos parámetros de conexión al CICS por IPIC
Ñ Creamos una SECTION IPICSERVER por cada servidor CICS al que queremos que
Ñ se pueda conectar el CTG
Ñ Sólo hay que ponerla si vamos a usar conexiones IPIC
SECTION IPICSERVER = CICS1         Ñ Nombre que le damos al servidor CICS
                                   Ñ Aparecerá cuando listemos los
                                   Ñ servidores al que podemos hacer
                                   Ñ peticiones desde el CTG
   description = Con. IPIC a CICS1 Ñ Descripción de la conexión
   hostname = pruebas              Ñ Dirección TCP/IP donde está el CICS
                                   Ñ Al estar ambos en la misma máquina
                                   Ñ sobre z/OS podemos usar su nombre
                                   Ñ Sino, tendríamos que usar la IP
   port = 20061                    Ñ Puerto donde más adelante definiremos
                                   Ñ el TCPIPSERVICE para conectar el CTG
                                   Ñ al CICS
   tcpkeepalive = Y                Ñ Con YES comprueba periódicamente que
                                   Ñ seguimos conectados al CICS
   srvidletimeout = 60             Ñ Tiempo en mins. que una conexión parada
                                   Ñ sigue abierta
   connecttimeout = 60             Ñ Tiempo en segs. que el CTG espera al
                                   Ñ CICS en un intento de establecer
                                   Ñ conexión
   sendsessions = 100              Ñ Número de transacciones o tareas CICS
                                   Ñ simultáneas sobre la conexión
                                   Ñ Establecido por este parámetro y por
                                   Ñ el parámetro RECEIVECOUNT de la
                                   Ñ definición de IPCONN (el que sea menor)
Ñ Los siguientes parámetros son opcionales y sirven para hacer comprobaciones
Ñ O se utilizan ambos a no se utiliza ninguno
Ñ Se utilizan en algunos casos cuando el CTG y el CICS no se encuentran en la
Ñ misma máquina lo que no es nuestro caso
Ñ  cicsapplid = CICSAPPL           Ñ APPLID del CICS
Ñ  cicsapplidqualifier = CICSQUAL  Ñ APPLID de la red donde está el CICS
ENDSECTION

Ñ Antes de la SECTION GATEWAY iría una sección que mapearía el nombre lógico
Ñ de un servidor CICS con su nombre actual, llamada SECTION LOGICALSERVER.
Ñ Como en nuestro sistema estos nombre coinciden no hace falta usar esta
Ñ sección

Ñ Esta sección define algunos parámetros de configuración del CTG.
Ñ Estos parámetros los podríamos establecer al arrancar el CTG o modificarlos
Ñ una vez arrancado desde el SD aunque normalmente no lo vamos a hacer.
Ñ Estos valores quedan invalidados por aquellos que demos desde el SD.
SECTION GATEWAY

Ñ Número de hilos iniciales que esperan conexiones desde el exterior
Ñ Por defecto es 1
Ñ initconnect=1

Ñ Número de hilos máximos para esperar conexiones desde el exterior
Ñ Por defecto es 100
```

```
Ñ maxconnect=100

Ñ Número de hilos iniciales para establecer conexión entre el CTG y el CICS
Ñ Por defecto es 1
Ñ initworker=1

Ñ Número de hilos máximos para establecer conexión entre el CTG y el CICS
Ñ Por defecto es 100
Ñ maxworker=100

Ñ Habilitar mensajes extra de traza (en caso de tener la traza activada).
Ñ trace=on

Ñ Deshabilita la entrada de comandos desde consola (SDSF).
Ñ Por defecto está habilitada la entrada de comandos.
Ñ Nos permitirá parar el CTG desde SDSF, listar estadísticas, etc.
Ñ noinput=on

Ñ Si está a on muestra en los mensajes la IP en vez nombres simbólicos usando
Ñ un servidor de DNS.
Ñ Mostrar los nombres (parámetro a off) puede afectar sustancialmente el
Ñ rendimiento por lo que al no ser estrictamente necesario lo dejamos a on.
nonames=on

Ñ Lo contrario del anterior.
Ñ No se especifica cual tiene preferencia pero lo dejamos deshabilitado
Ñ ya que es lo que queremos.
Ñ dnsnames=on

Ñ Suprime todas las salidas por consola (SDSF) (implica noinput=on)
Ñ No lo habilitamos ya que queremos que nos muestre cierta información
Ñ como errores, conexiones y desconexiones, etc.
Ñ quiet=on

Ñ Especifica la ruta Unix donde se va a guardar el fichero de traza
Ñ tfile=/tmp/ctg.trc

Ñ Establece el tamaño máximo en KB del fichero de traza.
Ñ Cuando se llega al máximo empieza a sobrescribir por el principio.
Ñ Por defeecto es 80 KB.
Ñ tfilesize=80

Ñ Tamaño máximo en bytes de cada bloque de datos de traza.
Ñ Por defecto es 80 bytes.
Ñ truncationsize=80

Ñ Desplazamiento en bytes de los datos antes de empezar a escribir la traza.
Ñ dumpoffset=0

Ñ Habilita el volcado de la pila de excepciones en la traza.
Ñ Por defecto a off.
Ñ stack=on

Ñ La siguiente sección habilita el Puerto que indiquemos (en este caso el
Ñ 2980) junto con algunos parámetros (conexiones máximas, timeout, etc.),
Ñ para recibir peticiones de estadísticas.
Ñ Si no vamos a utilizar esta funcionalidad, comentar las líneas
  protocol@statsapi.handler=com.ibm.ctg.server.RestrictedTCPHandler
  protocol@statsapi.parameters=connecttimeout=2000;\
  port=2980;\
  bind=;\
```

```
  maxconn=5;


Ñ Habilita el Puerto 2006 (por defecto) para recibir peticiones por TCP junto
Ñ con algunas características (timeouts, intervalo entre pings para comprobar
Ñ que el CTG sigue activo, etc.).
Ñ Comentar si no vamos a utilizar TCP.
protocol@tcp.handler=com.ibm.ctg.server.TCPHandler
protocol@tcp.parameters=port=2006;\
                        connecttimeout=2000;\
                        idletimeout=600000;\
                        pingfrequency=60000


Ñ Lo mismo que en el caso anterior pero para SSL.
Ñ Lo descomentaremos para probar el protocolo SSL.
Ñprotocol@ssl.handler=com.ibm.ctg.server.SslHandler
Ñprotocol@ssl.parameters=port=8050;\
Ñ                        connecttimeout=2000;\
Ñ                        idletimeout=600000;\
Ñ                        pingfrequency=60000;\
Ñ                        keyring=ServerKeyRing;\
Ñ                        keyringpw=default;\
Ñ                        clientauth=off;

Ñ Activa soporte para transacciones XA (si usamos conexiones EXCI)
Ñ En caso de conexiones IPIC está siempre activado, aunque no lo vamos a usar
Ñ xasupport=on

Ñ Validar LUW's
Ñ Sirve para que una LUW sólo pueda ser usada por la conexión que la crea
Ñ Por defecto está activado que es lo que queremos
Ñ uowvalidation=off

Ñ Indica en formato HHMMSS (horas, minutos y segundos) cuanto dura un
Ñ intervalo de estadísticas
Ñ De esta forma dura 3 horas
statint=030000

Ñ Indica en formato HHMMSS (horas, minutos y segundos) cuanto es el final del
Ñ día para las estadísticas
Ñ De esta forma acaba a las 00:00 horas
stateod=000000

Ñ Graba las estadísticas en SMF
Ñ Lo dejamos sin habilitar porque no nos hace falta
Ñ statsrecording=on

Ñ Envía estadísticas sobre comunicaciones a un balanceador de carga TCP/IP
Ñ No lo habilitamos al no tener un balanceador de carga
Ñ healthreporting=on

Ñ Lista de clases Java para usar como monitor de peticiones
Ñ No lo usamos
Ñ requestexits=com.ibm.ctg.samples.requestexit.NullMonitor

Ñ Clase que remapea los nombres de los servidores CICS en peticiones ECI
Ñ No lo usamos
Ñ cicsrequestexit=com.ibm.ctg.samples.ha.BasicCICSRequestExit

Ñ Tiempo en msg que pasa esperando a que se asigne un hilo de conexión entre
```

```
Ñ el CTG y el CICS. Si pasa este tiempo sin asignar un hilo se produce un
Ñ error. Por defecto se establece a 10000 msg. (10 seg)
Ñ workertimeout=10000

Ñ Tiempo en msg que pasa intentando finalizar peticiones una vez que se
Ñ cierra una conexión.
Ñ closetimeout=10000

Ñ Muestra las conexiones y desconexiones de las aplicaciones cliente
connectionlogging=on

Ñ Indica donde se muestran los datos de las conexiones y desconexiones.
Ñ De esta forma se muestran por la pantalla del SDSF.
log@info.dest=console
log@error.dest=console

ENDSECTION
```

Documento 2.1 Fichero ctg1a.ini de configuración del CTG

Con esto ya tendríamos preparado el archivo de inicialización del CTG. Esto es todo lo que tenemos que hacer en la parte Unix del mainframe.

2.3.2.3.- Miembro de variables de entorno

En esta paso vamos a ver como crear una librería, o data set PDS/E, para almacenar los diferentes miembros de variables de entorno del CTG que podamos tener. También crearemos un primer miembro de variables de entorno y lo configuraremos según nuestra instalación.

1.- Crear la librería para almacenar los miembros de variables de entorno. Para ello, desde la pantalla inicial de TSO tecleamos P.3.2 para acceder a la pantalla que se muestra en la Figura 2.9 desde donde podemos realizar diversas operaciones sobre data sets como crearlos (allocarlos), borrarlos, copiarlos, etc.

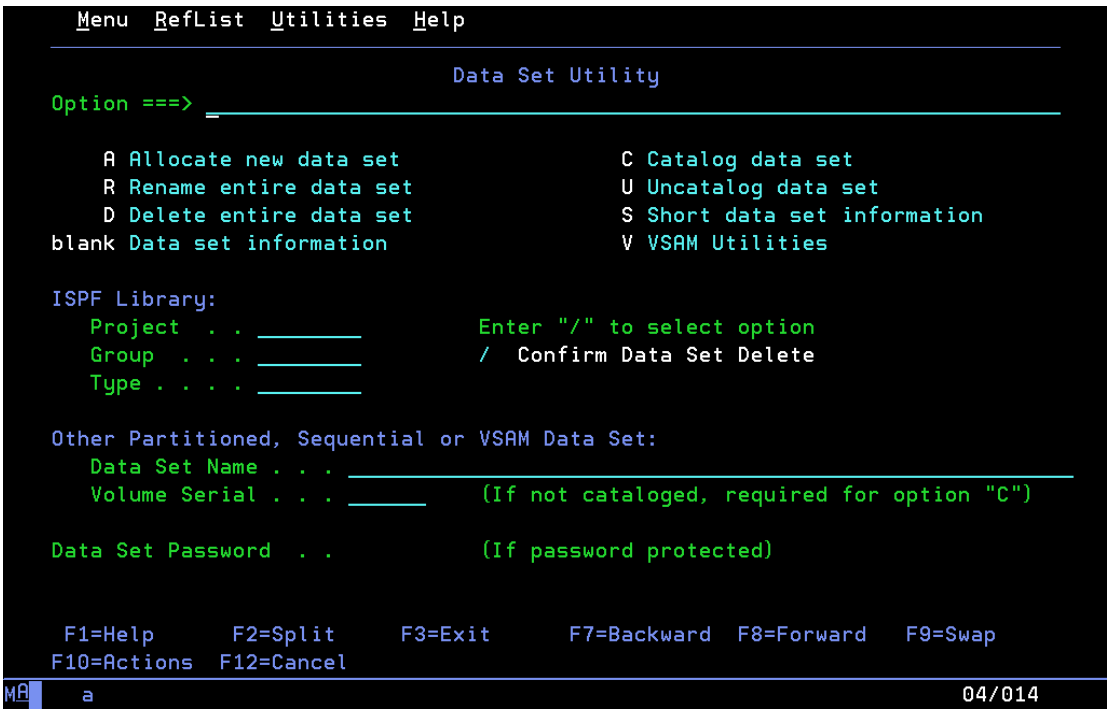


Figura 2.9 Pantalla P.3.4 para realizar operaciones sobre data sets

Si nos fijamos en la parte superior, si escribimos A en la línea de comandos podemos crear un nuevo data set. Para ello escribimos también el nombre que le queremos dar al data set que en este caso es CICSTG.V7R2M0.CTGENVAR.

La pantalla quedará como se puede ver en la Figura 2.10.

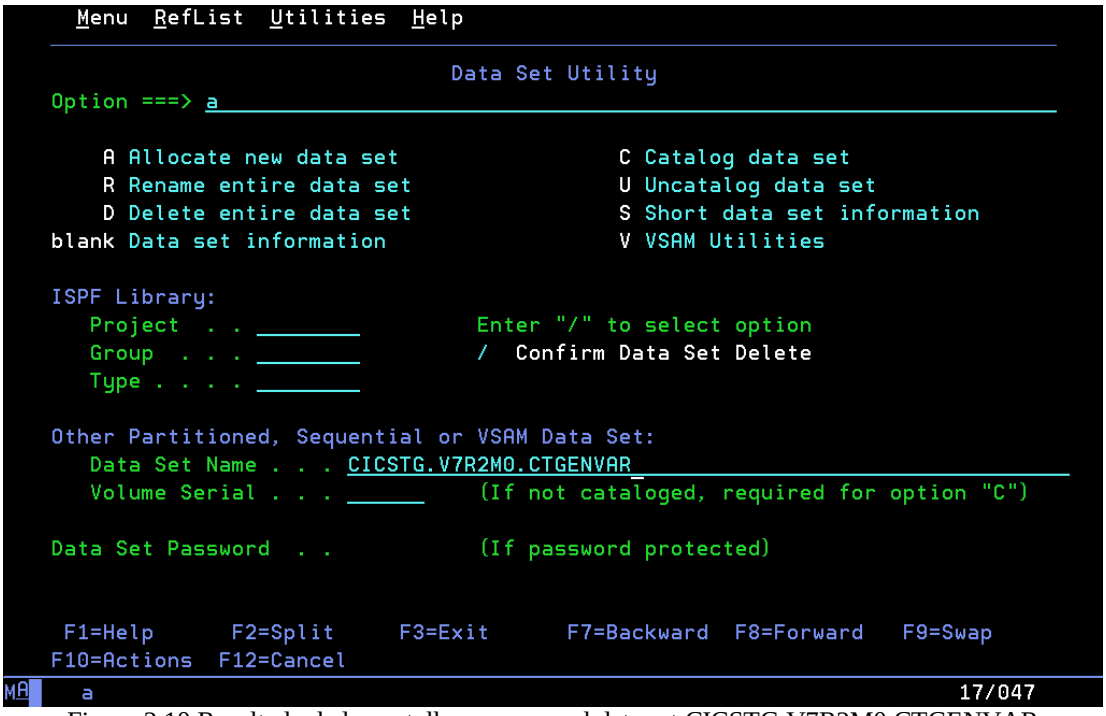


Figura 2.10 Resultado de la pantalla para crear el data set CICSTG.V7R2M0.CTGENVAR

Una vez hecho esto pulsamos Control, que en nuestro emulador simula al ENTER, para llegar a la pantalla mostrada en la Figura 2.11 donde podemos indicarle las propiedades del data set que vamos a crear que serán las que se muestran.

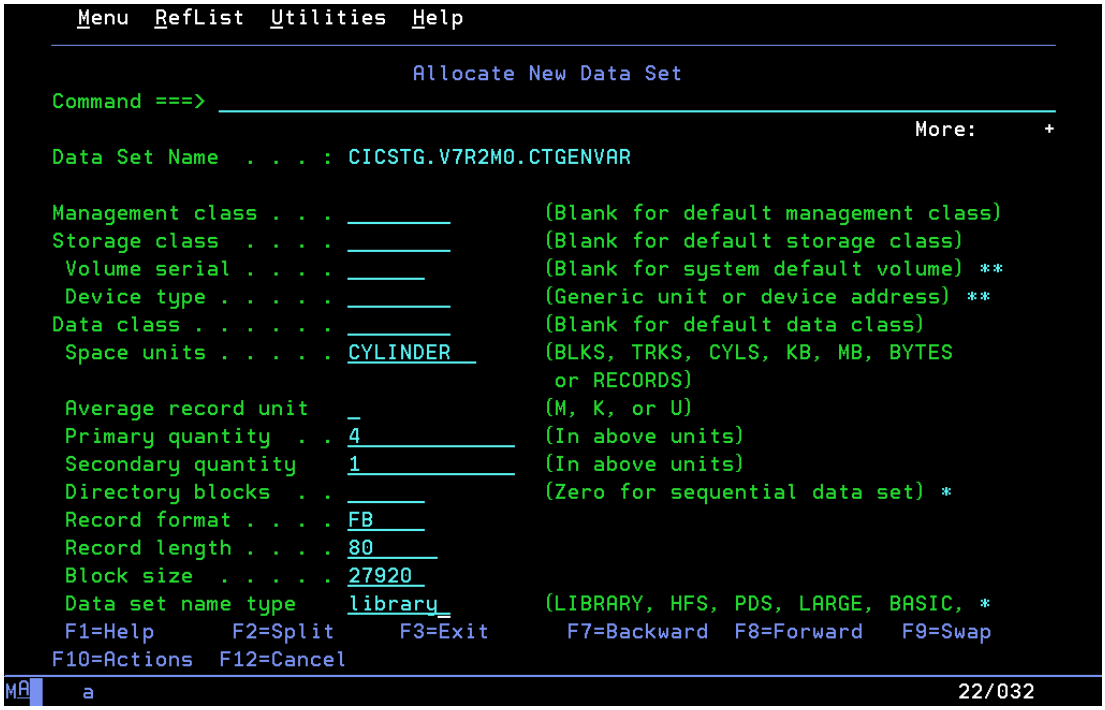


Figura 2.11 Propiedades del data set a crear

2.- Copiar el miembro siguiente a la librería que hemos creado:

CICSTG.V7R2M0.SCTGSAMP(CTGENV)

a

CICSTG.V7R2M0.CTGENVAR(ENVCTG1A)

Nos tenemos que acordar del data set y miembro a donde hemos hecho la copia ya que lo necesitaremos más adelante.

Para copiar un miembro de una librería, desde la pantalla de inicio de TSO tecleamos P.3.4 para dirigirnos a la pantalla que se muestra en la Figura 2.12 y escribir en "Dsname Level" el nombre del data set que queremos listar que en este caso será CICSTG.V7R2M0.SCTGSAMP.

```
 Menu  RefList  RefMode  Utilities  Help
 ------------------------------------------------------------------------------
                           Data Set List Utility
 Option ===> ____________________________________________________________________
                                                                        More:     +
    blank Display data set list              P Print data set list
        V Display VTOC information          PV Print VTOC information

 Enter one or both of the parameters below:
    Dsname Level . . . CICSTG.V7R2M0.SCTGSAMP_____________________
    Volume serial  . . ______

 Data set list options
    Initial View . . . 1  1. Volume       Enter "/" to select option
                          2. Space        /  Confirm Data Set Delete
                          3. Attrib       /  Confirm Member Delete
                          4. Total        /  Include Additional Qualifiers
                                          /  Display Catalog Name

 When the data set list is displayed, enter either:
   "/" on the data set list command field for the command prompt pop-up,
   an ISPF line command, the name of a TSO command, CLIST, or REXX exec, or
  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=Actions  F12=Cancel
MA    a                                                          10/046
```

Figura 2.12 Pantalla para listar data sets

Una vez pulsado Control nos lleva a la pantalla de la Figura 2.13. En esta podemos ver un listado de los data sets que coinciden con el nombre buscado. En este caso sólo hay uno. Introducimos una "v" delante del nombre del data set para listar los miembros que contiene. Si quisiéramos editar alguno de los miembros del data ser, introduciríamos delante del nombre del data set una "e" en vez de una "b".

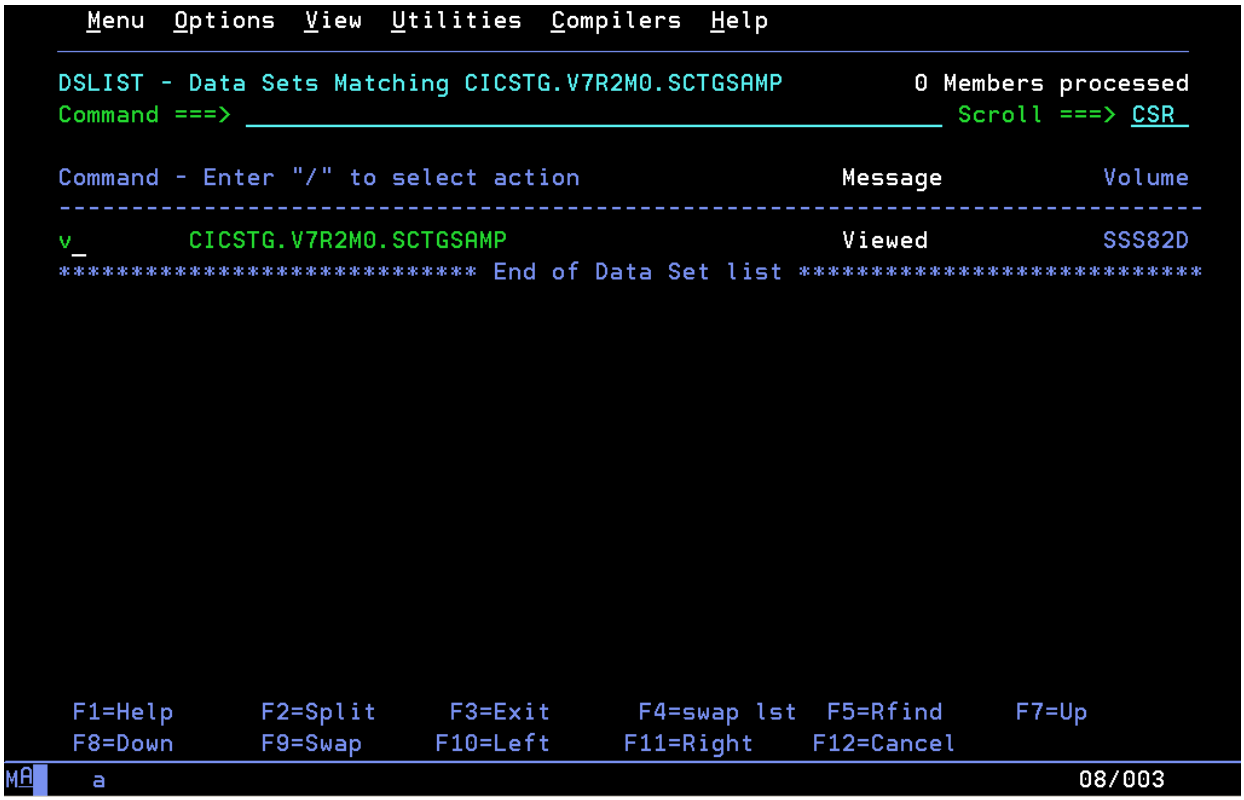


Figura 2.13 Listado de data sets

Una vez pulsado Control nos llevará a la pantalla de la Figura 2.14 donde se muestran los miembros que contiene este data set. Ahora nos situamos con el cursor delante del miembro que queremos copiar y escribimos una “c” delante de él.

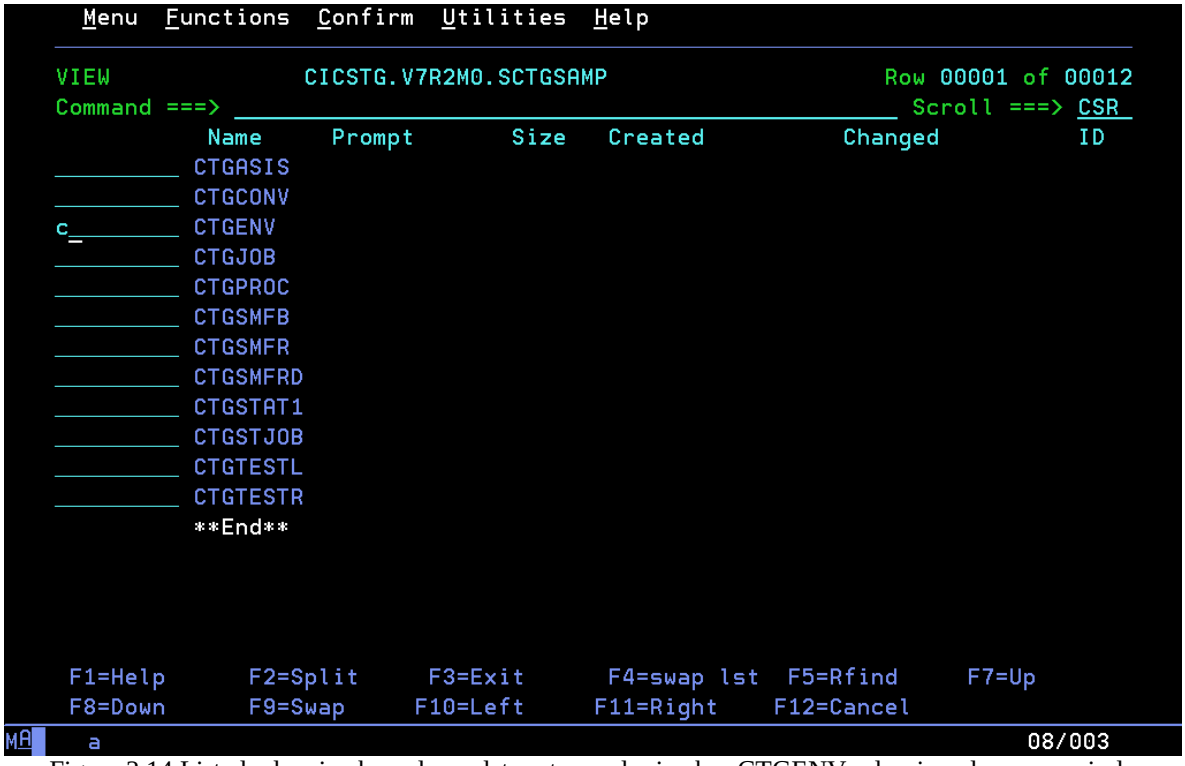


Figura 2.14 Listado de miembros de un data set con el miembro CTGENV seleccionado para copiarlo

Pulsamos Control con lo que nos aparece la pantalla de la Figura 2.15 donde podemos introducir ciertos parámetros para copiar el miembro.

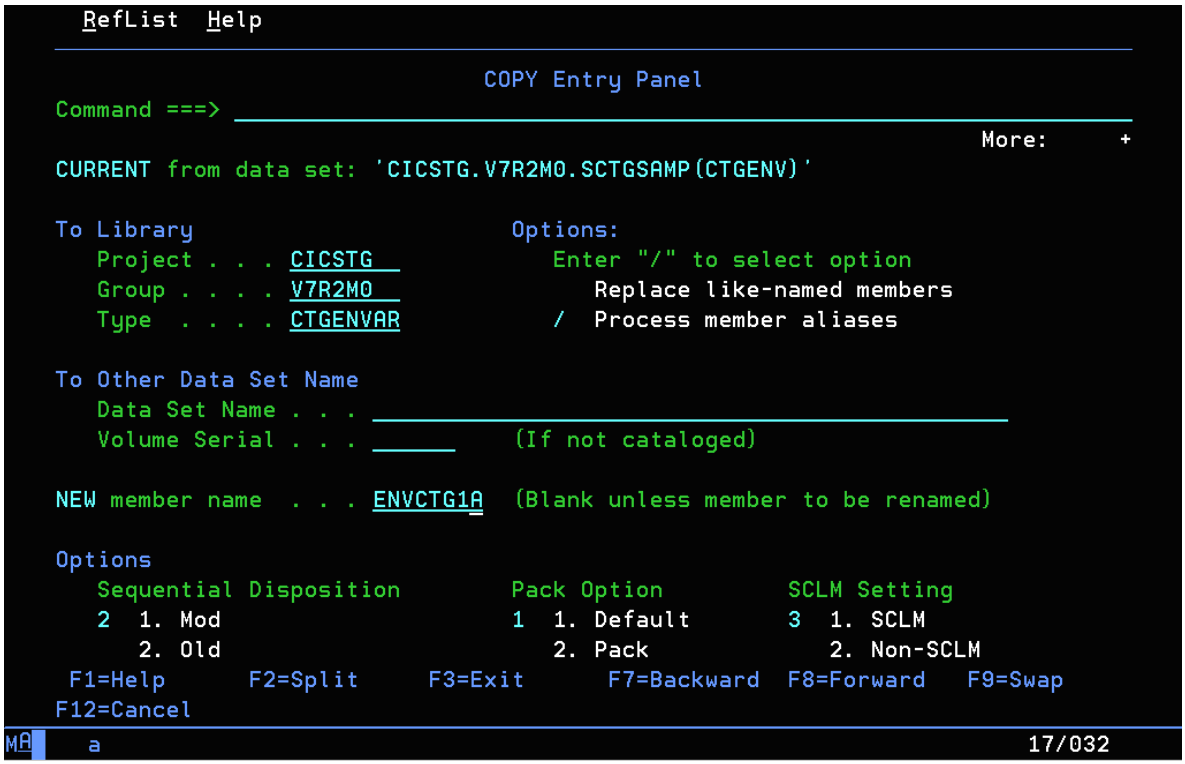


Figura 2.15 Panel para copiar de un miembro

A nosotros nos interesan los campos “To Library” donde introducimos el nombre del data set donde queremos copiar el miembro, que en este caso es el CICSTG.V7R2M0.CTGENVAR. En “NEW member name” le damos el nombre con el que queremos que se copie el miembro, en caso de que no queramos que sea el mismo que el miembro original, que en este caso será ENVCTG1A. Pulsamos Control con lo que ya tenemos copiado el miembro.

3.- Una vez copiado lo adaptamos a nuestra instalación definiendo los siguientes valores para las variables que hay en el miembro.

Para poder editar el miembro, desde la pantalla principal de TSO tecleamos P.3.4 para poder buscar el data set que queramos como hemos visto en la Figura 2.12. Buscamos el data set CICSTG.V7R2M0.CTGENVAR. Ahora, en la pantalla de la Figura 2.13 en vez de introducir una “v” delante del data set introducimos una “e” para poder modificar sus miembros además de listarlos.

Situamos el cursor delante del miembro que queremos modificar y pulsamos Control. De esta forma se abre el miembro para que lo podamos consultar y editar, como se muestra en la Figura 2.16.

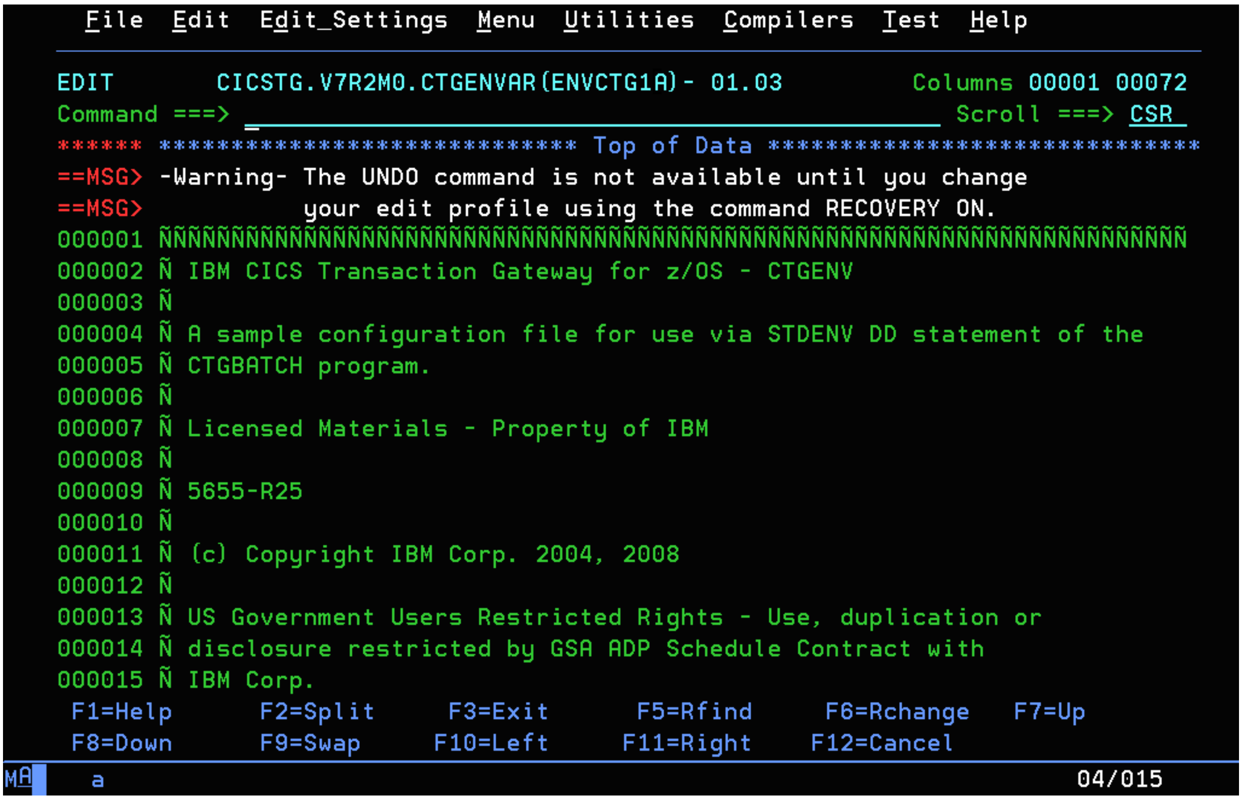


Figura 2.16 Miembro ENVCTG1A abierto para consulta y edición

Editamos el miembro para quede de la siguiente manera. Como cuando hemos editado un archivo Unix, las líneas que empiezan por Ñ son comentarios. En el Documento 2.2 podemos ver como queda el miembro con las variables de entorno para nuestra instalación

```
Ñ Ruta Unix donde está el fichero .ini que hemos modificado anteriormente que
Ñ es el que vamos a usar para inicializar este CTG
CICSCLI=/etc/ctgvar/ctg1a.ini

Ñ Ruta Unix donde se encuentran los archivos mínimos para poder ejecutar
Ñ el ctgstart (arranque del CTG). Son las utilidades básicas de Unix que
Ñ normalmente se encuentran en /bin y la ruta bin donde está instalado Java
PATH=/bin:/usr/lpp/java/J5.0/bin

Ñ Directorio temporal necesario para iniciar el CTG
TMPDIR=/tmp

Ñ Ejecuta los daemons del CTG en el mismo espacio de direcciones que
Ñ el CTG en si mismo. Recomendado
_BPX_SHAREAS=YES

Ñ Nombre del recurso CONNECTION definido en el CICS. Necesario sólo si
Ñ queremos conexión por EXCI
Ñ Cuando usemos conexiones EXCI, descomentamos esta línea
ÑDFHJVPIPE=CICSTG1A

Ñ Nombre de los servidores CICS a los que queremos conectarnos por EXCI
Ñ Creamos una entrada DFHJVSYSTEM_XX por cada CICS al que nos queremos
Ñ conectar por EXCI
Ñ Cuando usemos conexiones EXCI, descomentamos la primera línea
ÑDFHJVSYSTEM_00=CICS1 – Conexión EXCI a CICS 1
ÑDFHJVSYSTEM_01=<APPLID_Z>-Example production server
```

```
Ñ Indica si vamos a usar EXCI y el data set donde se encuentran los
Ñ módulos necesarios
CTG_EXCI_INIT=NO
ÑSTEPLIB=CICSTS.V3R2M0.CICS.SDFHEXCI

Ñ Determina como los daemons del CTG van a reusar las pipes EXCI
Ñ Por defecto está a ALL
CTG_PIPE_REUSE=ALL

Ñ Especifica el nombre del recurso que va a gestionar una instancia en
Ñ particular del CTG
Ñ Por defecto le damos el nombre del CTG y el sufijo .IBM.UA que tiene
Ñ que llevar obligatoriamente
CTG_RRMNAME=CTG1A.IBM.UA

Ñ Especifica el máximo número de transacciones XA concurrentes
Ñ Sólo hay que especificarlo si tenemos xasupport=on
ÑCTG_XA_MAX_TRAN=1000

Ñ Indica si usamos autentificación de usuario/contraseña por EXCI
AUTH_USERID_PASSWORD=NO

Ñ Indica si en las contraseñas se puede usar mayúsculas y minúsculas
Ñ Si está a YES podemos usar ambas y si está a NO convierte todo a mayúsculas
Ñ Sólo hay que indicarlo se tenemos AUTH_USERID_PASSWORD=YES
ÑCTG_MIXEDCASE_PW="NO"

Ñ Indica si el CTG puede ser swappable
Ñ Swappable significa que si el sistema se queda sin memoria principal
Ñ aquellos programas marcados como swappables y que no estén ejecutandose
Ñ pueden ser sacados a memoria secundaria hasta que se los necesite
Ñ Se recomienda no ponerlo, así el CTG funciona en modo no swappable, a no
Ñ ser que el CTG funcione en Modo Local
Ñ Si está como swappable y a sido sacado a memoria secundaria, cualquier
Ñ petición que se haga al CTG fallará
ÑCTG_SWAPPABLE=NO

Ñ Diferentes parámetros que se puede usar en el arranque del CTG
ÑCTGSTART_OPTS=-x -j-Dgateway.T.setTFile=<trace path>/ctgtrace.txt \
Ñ-j-Dgateway.T.setJNITFile=<trace output path>/jnitrace.txt

Ñ Longitud de los mensajes que devuelve el CTG
Ñ Ponemos 80 ya que es el ancho que muestra nuestra configuración de SDSF
COLUMNS=80

Ñ Ajuste para reflejar la zona horaria en la que se encuentra el CTG
Ñ Para más información consultar SA22-7802 z/OS UNIX System Services Command
Ñ Reference
Ñ La que se muestra es válida para la zona horaria de Madrid
TZ=CET-1CEST,M3.5.0/2,M10.5.0/3

Ñ Directorio para volcar diferentes volcados de memoria en caso de fallo
Ñ_CEE_DMPTARG=/tmp/jvmdump
```

Documento 2.2 Miembro ENVCTG1A con las variables de entorno para nuestro CTG

Con esto ya tendríamos configurado el miembro de variables de entorno de acuerdo a nuestros requerimientos.

2.3.2.4.- Procedimiento de arranque

El siguiente paso es crear un JCL de procedimiento de arranque que ejecutaremos como un job para que se inicie el CTG.

1.- Copiamos el procedimiento de arranque de ejemplo de

CICSTG.V7R2M0.SCTGSAMP(CTGPROC)

a

PRUEBAS.USER.PROCLIB(CTG1A)

El nombre del miembro a donde lo copiamos puede ser cualquiera pero lo dejamos con el mismo nombre que el CTG para que sea más fácil recordarlo.

2.- Modificamos el procedimiento de arranque para que quede de la siguiente manera y se adapte a nuestra instalación. En este caso, las líneas con comentario son las que empiezan por //*. En el Documento 2.3 podemos ver como queda el procedimiento de arranque para nuestro CTG.

```
//CTGPROC  PROC
//* Ruta Unix donde está instalado el CTG
// SET CTGHOME='/usr/lpp/cicstg/v7r2m0/'
//* Cualificador de los data sets donde se encuentra instalado el CTG
// SET CTGHLQ='CICSTG.V7R2M0'
//* Data set y miembro donde se encuentran las variables de entorno
//* Indicamos el miembro de variables de entorno que acabamos de personalizar
// SET CTGUSR='CICSTG.V7R2M0.CTGENVAR(ENVCTG1A)'
//* Opciones de ejecución acabadas en /
//* No utilizamos ninguna
// SET LEOPTS='/'
//*
//* Opciones de arranque del CTG
//* REGION indica cuanta memoria puede coger el CTG para funcionar.
//* 0M indica que infinita aunque luego esto lo cambiaremos
//* TIME indica el tiempo máximo de CPU por día que puede usar el job
//* 1440 es sinónimo de 24 horas, lo que es similar a infinito
//CTG      EXEC PGM=CTGBATCH,REGION=0M,TIME=1440,
//         PARM='&LEOPTS.&CTGHOME./bin/ctgstart -noinput '
//STEPLIB  DD DSN=&CTGHLQ..SCTGLOAD,DISP=SHR
//*
//* Lugar donde se muestran los mensajes de salida y de error
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD DSN=&CTGUSR.,DISP=SHR
//*
//PEND
//
```

Documento 2.3 Procedimiento de arranque para nuestro CTG

Con esto ya tendríamos preparado el procedimiento de arranque que necesitamos para iniciar el CTG. Los siguientes pasos son crear las definiciones de RACF, o definiciones de seguridad, necesarias para que el CTG funcione.

2.3.2.5.- Definiciones de RACF

El siguiente paso en la instalación del CTG es definir en RACF la *Started Class*, para que el CTG tenga permisos para ejecutarse, y el usuario al que va esta ligado la *Started Class*.

El nombre de la *Started Class* será el que le hayamos dado al procedimiento de arranque, CTG1A, y al usuario lo vamos a definir con el nombre SCCTG, de StartedClassCTG.

1.- Creamos en Unix el directorio home con permisos 755 para el usuario al que va a estar ligado la *Started Class*. El directorio home estará en "/u/SCCTG".

2.- Tenemos que crear las siguientes definiciones de RACF:

- Un usuario Unix
- Una *Started Class*
  - No definimos la *Started Class* CTG1A sino la CTG1* de forma que más adelante podamos tener la *Started Class* CTG1B, etc. para un grupo de CTGs en *Port Shared* sin tener que volver a hacer definiciones
- Asociar el usuario a la *Started Class*
- Para hacer el CTG no swappable, debemos incluir al usuario en la lista de acceso del perfil BPX.STOR.SWAP de la clase FACILITY con permiso de lectura

Esto lo haremos ejecutando el JCL que podemos ver en el Documento 2.4.

```
//RACFCTG  JOB (77005),'JPM',
//      CLASS=I,MSGCLASS=X,MSGLEVEL=(1,1),
//      NOTIFY=&SYSUID
//S1       EXEC PGM=IKJEFT01,REGION=1M
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD * BPX.SMF
//* DEFINICIONES PARA CTG: STARTED CLASS Y USUARIO
   ADDUSER SCCTG DFLTGRP(STCUSR )                                  +
     DATA('USUARIO STARTED CLASS CTG                               ') +
     OMVS(UID(0) HOME('/u/SCCTG') PROGRAM('/bin/sh')) NOPASSWORD
   RDEFINE STARTED CTG1*.* STDATA( USER(SCCTG) PRIVILEGED(NO) +
     TRUSTED(NO) TRACE(NO) )
   PERMIT BPX.STOR.SWAP CLASS(FACILITY) ACCESS(READ) ID(SCCTG)
   SETROPTS RACLIST(FACILITY) REFRESH
   SETROPTS RACLIST(STARTED) REFRESH
/*
```

Documento 2.4 JCL con definiciones de RACF

En el caso de usar EXCI, debemos añadir los siguientes permisos:

```
RDEFINE SURROGAT *.DFHEXCI UACC(NONE) OWNER(SYSADMIN)
PERMIT *.DFHEXCI CLASS(SURROGAT) ID(SCCTG) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

Para definir la FACILITY en caso de que no lo estuviera, añadiremos además lo siguiente antes de "PERMIT BPX.STOR.SWAP …":

```
RDEFINE FACILITY BPX.STOR.SWAP
SETR RACLIST(FACILITY) REFRESH
```

Con esto ya hemos realizado todas las operaciones necesarias sobre TSO. El siguiente paso es crear e instalar en CICS las definiciones necesarias tanto si vamos a usar una conexión EXCI como una IPIC.

2.3.3.- Creación de la conexión entre el CTG y el CICS

Para poder definir recursos en el CICS e instalarlos, lo primero que tenemos que hacer una vez que nos hemos conectado a CICS correspondiente, en este caso el CICS1, por medio del emulador, es identificarnos en el sistema. Para ello tecleamos CESN y pulsamos Control. Esto arranca la transacción CESN que, como ya hemos indicado, es la transacción suministrada por CICS para identificarnos en el sistema. En algunos sistemas esto no hará falta ya que por defecto se arranca la CESN al conectarnos al CICS.

En la Figura 2.17 podemos ver la pantalla inicial de la CESN donde tenemos que introducir nuestro usuario y contraseña y pulsar Control para identificarnos.

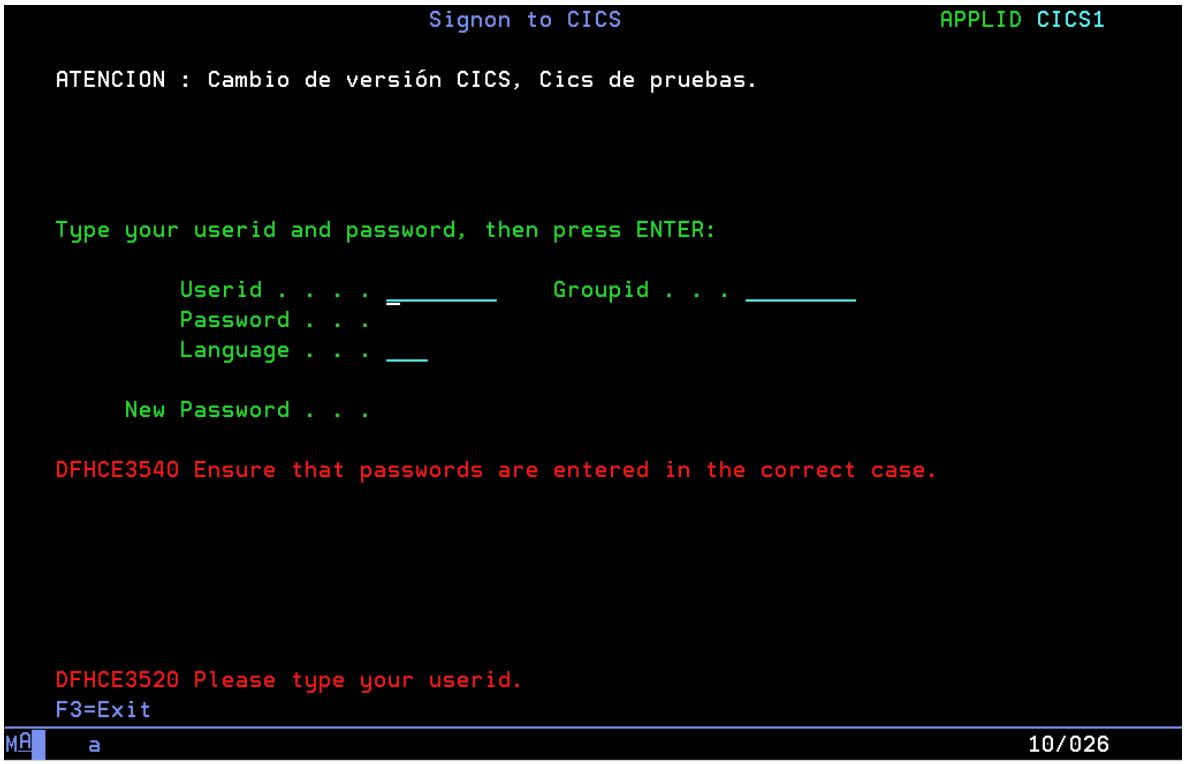


Figura 2.17 Pantalla inicial de la CESN

Una vez que nos hayamos identificado correctamente, en la parte inferior de la pantalla nos aparecerá el mensaje que podemos ver en la Figura 2.18.

Figura 2.18 Mensaje de identificación correcta

Con esto ya nos hemos identificado correctamente y podemos seguir definiendo los recursos necesarios e instalándolos.

2.3.3.1.- Conexiones EXCI

Antes de realizar las definiciones necesarias para la conexión EXCI, tenemos que comprobar que en la SIT, o System Initialization Table, del CICS correspondiente, tenemos los siguientes parámetros con los valores adecuados:

- ISC = YES
- IRCSTRT = YES

Para comprobar estos valores, por medio de una sesión TSO tenemos que comprobar el data set y el miembro que actúa como SIT para el CICS donde nos queremos conectar. En nuestro caso, el data set es el PRUEBAS.CICS1.TABLAS y el miembro es el DFHSITJT.

Comprobando este miembro vemos que tenemos los parámetros antes mencionados con el valor adecuado. En caso de no ser así, habría que modificar el miembro y volver a compilar la SIT lo que no vamos a explicar como hacerlo por considerar que queda fuera del ámbito del proyecto.

Lo primero que tenemos que hacer para poder usar conexiones EXCI es descomentar las siguientes líneas en el miembro de las variables de entorno:

- DFHJVPIPE=CICSTG1A
- DFHJVSYSTEM_00=CICS1 – Conexión EXCI a CICS 1

Para crear una conexión EXCI entre el CTG y el CICS, tenemos que definir e instalar los siguientes recursos:

- CONNECTION
- SESSIONS

Los recursos tienen que ir definidos en un grupo con nombre de 8 caracteres máximos que en este caso le llamaremos EXCTG1A, de EXCI y CTG1A.

A las CONNECTION y SESSIONS la llamaremos CTG1 ya que su nombre sólo puede ser de 4 caracteres. No hay ningún problema por definir dos recursos de distinto tipo con el mismo nombre.

Una vez que nos hemos identificado en el CICS, ejecutamos el comando CEDA DEF CONNECTION(CTG1) G(EXCTG1A) como podemos ver en la Figura 2.19.

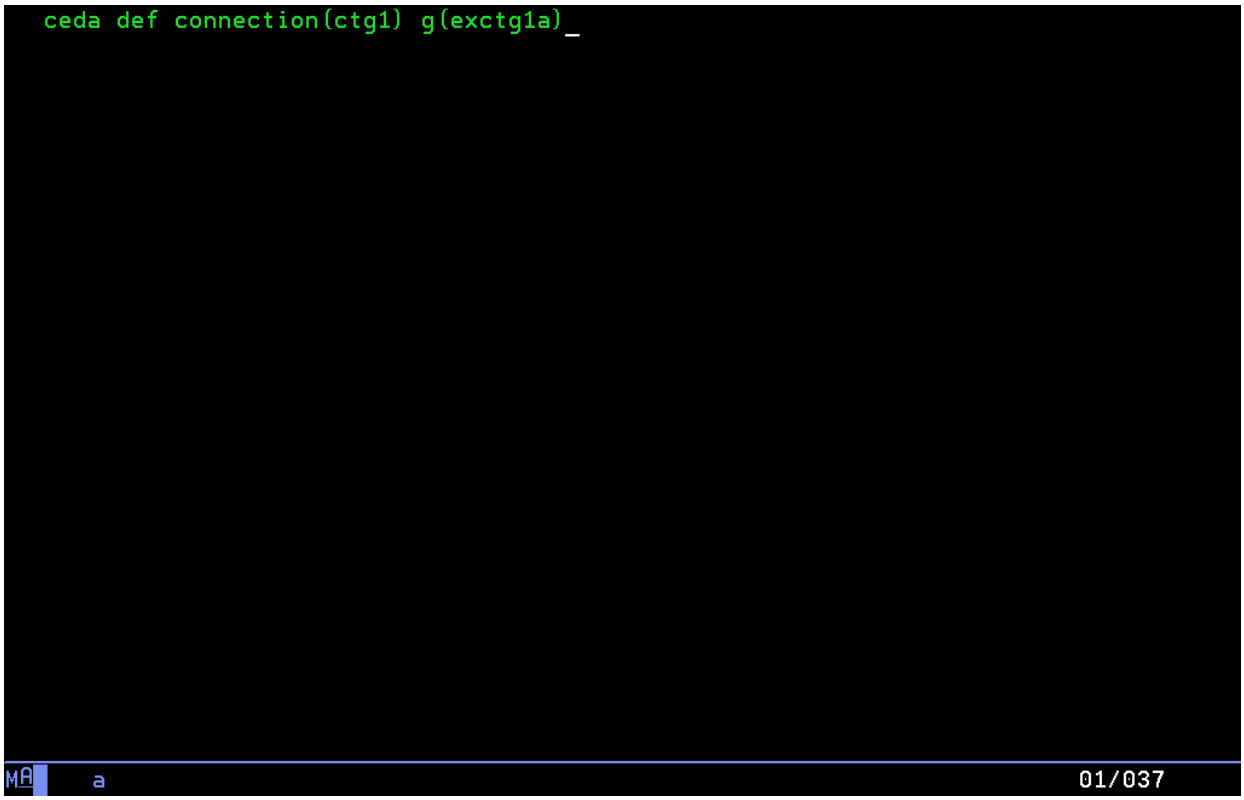


Figura 2.19 Comando para definir una CONNECTION

Nos aparecerá una pantalla como la que podemos ver en la Figura 2.20.

Figura 2.20 Pantalla de definición de CONNECTION

Vemos como en la parte inferior nos indica que el grupo se ha creado, ya que no existía.

Por medio de las teclas F7 y F8 nos desplazamos adelante y atrás en la pantalla de definición de la CONNECTION. La mayoría de los atributos los dejamos como están por defecto pero a algunos los modificamos para darles los siguientes valores:

- CONnection : CTG1
- Group : EXCTG1A
- Netname : CTG1A
  - El applid del CTG
- ACcessmethod : IRc
- Protocol : Exci
- Conntype : Specific
- ATtachsec : Identify

Una vez que tengamos todos los parámetros adecuados pulsamos Control para terminar con la definición de la CONNECTION. En la parte inferior de la pantalla nos aparecerá el mensaje que podemos ver en la Figura 2.21.

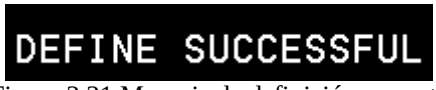


Figura 2.21 Mensaje de definición correcta

Una vez que hemos terminado con la definición de la CONNECTION, pulsamos F3 para salir de la pantalla de definición y luego la tecla Pausa para borrar la pantalla y poder empezar a definir la SESSIONS.

Para definir una SESSIONS de nombre CTG1 en el grupo EXCTG1A ejecutamos el comando CEDA DEF SESSIONS(CTG1) G(EXCTG1A).

Nos aparecerá una pantalla similar a la de la definición de la CONNECTION pero con diferentes parámetros.

Los parámetros a los que ahora tenemos que cambiar su valor son los siguientes:

- Sessions : CTG1
- Group : EXCTG1A
- Connection : CTG1A
- Protocol : Exci
- RECEIVEPfx : S1
- RECEIVECount : 004
- SENDSize : 30720
- RECEIVESize : 30720

Cuando tengamos los parámetros con sus valores correctos pulsamos Control y nos aparecerá el mensaje de la Figura 2.21. Pulsamos F3 y Pausa para salir de la definición y borrar la pantalla. Con esto ya tenemos definidos los recursos necesarios.

Ahora tenemos que instalar el grupo con sus definiciones. Esto lo hacemos ejecutando el comando CEDA I G(EXCTG1A).

Nos aparecerá en la parte inferior de la pantalla el mensaje de la Figura 2.22 indicándonos que la instalación del grupo se ha realizado correctamente.

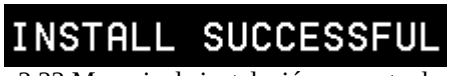


Figura 2.22 Mensaje de instalación correcta de grupo

Si acabamos aquí, cada vez que se pare y arranque el CICS tendríamos que instalar el grupo. Para evitar esto, podemos añadir el grupo a una de las listas de arranque de CICS. De esta forma, al arrancar el CICS automáticamente se instalan todos los recursos que se encuentran en los grupos que están en las listas de arranque. En nuestro caso, una de estas listas es la LISTAARR. Para añadir el grupo EXCTG1A a la lista ejecutamos el comando CEDA ADD GROUP(EXCTG1A) LIST(LISTAARR).

Con esto ya tendríamos todo lo necesario para que el CTG funcione por medio de conexiones

2.3.2.- Conexiones IPIC

Para poder conectarnos por IPIC al CICS, tenemos que tener definidos los siguientes recursos:

- TSPIPSERVICE
- IPCONN

El primero es obligatorio definirlo mientras que el segundo podemos definirlo a mano o hacer que se defina e instale automáticamente. En principio utilizaremos esta segunda opción ya que, de momento, la definición que hace del IPCONN por defecto nos es válida. Lo definiremos manualmente más adelante cuando queramos habilitar la seguridad.

En este caso el nombre del TCPIPSERVICE puede ser de 8 caracteres por lo que le daremos el nombre CTG1A. El grupo donde definimos los recursos para la conexión IPIC será el IPCTG1A.

Para definir el TSPIPSERVICE, ejecutamos el comando CEDA DEF TCPIPSERVICE(CTG1A) G(IPCTG1A). Nos aparecerá una pantalla similar a la de la Figura 2.20 pero ahora los parámetros que tenemos que modificar y su valor son los siguientes:

- Tcpipservice : CTG1A
- Group : IPCTG1A
- POrtnumber : 20061
  - Puerto reservado en el CICS para conectar el CTG1A
- PROtocol : IPIC
- TRansaction : CISS

Cuando hayamos terminado pulsamos Control y nos aparecerá el mensaje de la Figura 2.21. Con esto ya tenemos definidos los recursos necesarios para poder usar una conexión IPIC.

Instalamos el grupo con el comando CEDA I G(IPCTG1A). Nos aparecerá el mensaje de la Figura 2.22 con lo que ya tenemos todo preparado para poder usar conexiones IPIC.

Al igual que con la conexión EXCI, si queremos que automáticamente se instale el grupo al arrancar el CICS, añadiremos el grupo IPCTG1A a la lista LISTAARR con el comando CEDA ADD GROUP(EPCTG1A) LIST(LISTAARR).

2.3.4.- Arranque y parada del CTG

2.3.4.1.- Arranque del CTG

Siguiendo los pasos vistos anteriormente ya tenemos todo preparado para poder iniciar el CTG. El CTG necesita una serie de subsistemas o productos pare funcionar que tienen que estar

inicializados antes de iniciar la ejecución del CTG. Estos productos y su orden de arranque son los siguientes:

1. RRS, se inicia al arrancar el mainframe o la LPAR
2. TCP/IP
3. CICS
4. CTG

Para poder iniciar el CTG, desde la pantalla inicial de TSO tecleamos SD y pulsamos Control para iniciar el SDSF. Una vez en la pantalla inicial del SDSF, tecleamos LOG y pulsamos Control para abrir el Log del sistema. En este Log podemos ver mensajes sacados por los programas en ejecución, comandos ejecutados, mensajes de respuesta del z/OS, etc. Esto lo hace idóneo para iniciar el arranque del CTG y ver si este se realiza correctamente. Podríamos arrancar el CTG desde cualquier cola del SDSF utilizando el mismo método que vamos a ver a continuación pero sería más complicado comprobar su arranque correcto.

Para iniciar el arranque del CTG, nos situamos como ya hemos dicho en el panel de Log del SDSF que podemos ver en la Figura 2.23.

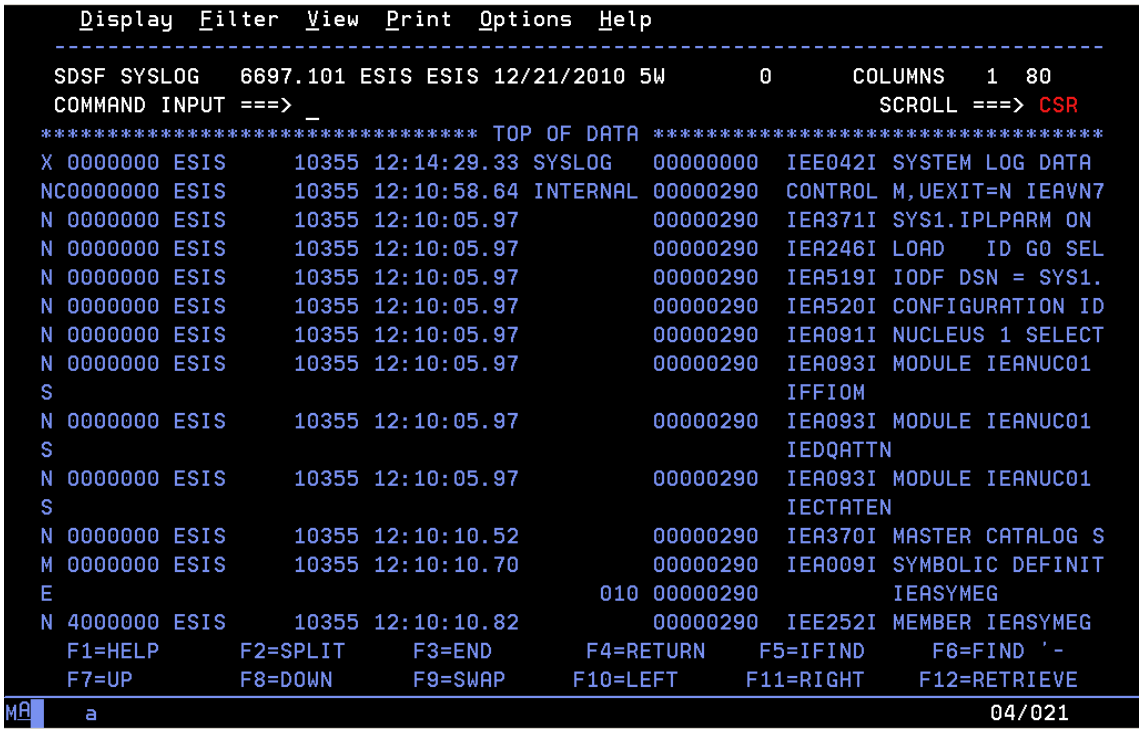


Figura 2.23 Panel de Log del SDSF

Una vez en este panel pulsamos repetidamente F8 hasta llegar a la parte inferior del Log. Una vez ahí, para arrancar el CTG ejecutamos desde la línea de comandos del panel el comando "/s <*nombre_procedimiento_arranque*>", sin las comillas, lo que en nuestro caso será "/s ctg1a".

Si pulsamos repetidamente F8, veremos que en el Log empezarán a aparecer mensajes que nos indica que el CTG se está arrancando. Sabremos que el CTG ha arrancado correctamente y de forma satisfactoria cuando en el Log veamos que aparece el mensaje que podemos ver en la Figura 2.24.

```
CTG6512I CTG1A    CICS TRANSACTION GATEWAY INITIALIZATION COMPLETE
```

Figura 2.24 Mensaje de inicialización completa del CTG

Con esto ya tendríamos el CTG preparado para recibir y contestar peticiones.

2.3.4.2.- Parada del CTG

En algunos casos, por ejemplo porque queremos realizar modificaciones o porque queremos apagar el mainframe, vamos a necesitar parar el CTG. Para ello nos situaremos al final del panel de Log como hemos visto anteriormente y ejecutaremos alguno de los siguientes comandos:

- /p <*nombre_job*>
- /f <*nombre_job*>,appl=shutdown
- /f <*nombre_job*>,appl=immediate

Los dos primeros comandos realizan una parada normal del CTG. El CTG espera a que las transacciones que se están ejecutando finalicen aunque no admite más peticiones. Se recomienda parar el CTG con alguno de estos comandos.

El tercer comando realiza una parada inmediata del CTG. Las transacciones que se estaban ejecutando finalizan de manera anormal, Abnormally END o ABEND para el mainframe. Si se estaba ejecutando alguna LUW, esta finaliza y se hace *backout* de los cambios que había realizado. Si se estaba ejecutando alguna transacción XA, se hace *backout* de los cambios que había realizado y las tareas que quedan por realizar se quedan en espera en el RRS hasta que el CTG se reinicia, o se ejecutan en otro CTG del mismo grupo.

Cuando lanzamos un comando de parada del CTG, sabemos que este ha parado correctamente cuando en el Log se nos muestra el mensaje que podemos ver en la Figura 2.25.

Figura 2.25 Mensaje de parada correcta del CTG

El orden de parada de los subsistemas o productos que hemos visto anteriormente que son necesarios para que funcione el CTG es el inverso al de arranque.

2.4.- Prueba con COMMAREA

Una vez que ya tenemos el CTG configurado y listo para atender peticiones, el siguiente paso es ver como realizar estas peticiones. Para ello vamos mostrar la manera de hacerlo primero desde aplicaciones escritas en Java y luego desde aplicaciones escritas en .Net.

Como ya hemos comentado, sólo vamos a utilizar la API ECI ya que tanto la EPI como la ESI no están disponibles debido al tipo de arquitectura que tenemos.

2.4.1.- Aplicaciones Java

A la hora de hacer peticiones Java de programas CICS hay dos clases fundamentales que vamos a utilizar:

- JavaGateway
- ECIRequest

La primera de ellas la utilizaremos para realizar la conexión entre nuestra aplicación y el CTG. Será por donde enviemos y recibamos las peticiones.

La segunda la vamos a utilizar para enviar los datos de la petición que vamos a realizar, usuario y contraseña en caso de ser necesario, programa que queremos ejecutar, datos de entrada, etc. También es por donde vamos a recibir la respuesta del CTG con los resultados obtenidos.

Estas clases se encuentran ambas en la librería ctgclient.jar que se provee junto con el CTG. Dentro de esta librería, ambas clases se encuentran dentro del paquete com.ibm.ctg.client. Este paquete será el que tengamos que importar a nuestras aplicaciones para hacer uso de ambas clases.

Para añadir una librería a un proyecto de NetBeans, que es la *suite* de programación que vamos a usar, tenemos que hacer lo siguiente:

1. Descargarnos a nuestro ordenador, en modo binario, desde la parte Unix del mainframe el archivo ctgclient.jar que en general se encontrará en la ruta /usr/lpp/cicstg/<*version_ctg*>/classes
   - Para ello nos conectaremos por FTP al mainframe con algún programa que lo permite como el BlueZone Secure FTP v5.0 que se ha usado en nuestro caso y trasferiremos el archivo. Las instrucciones de cómo hacer esto las tenemos en el Anexo 1.
2. Una vez creado el proyecto de NetBeans donde queremos importar la librería, importamos el archivo a la carpeta “Bibliotecas”.
   - Hacemos clic derecho sobre la carpeta “Bibliotecas” y seleccionamos “Agregar archivo JAR/carpeta…” como podemos ver en la Figura 2.26.

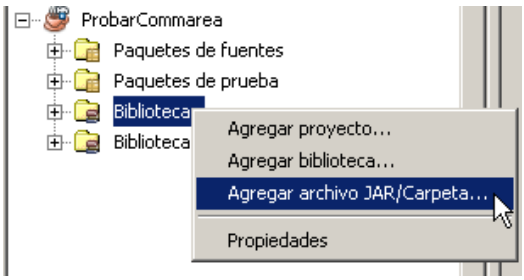


Figura 2.26 Agregar archivo JAR/carpeta…

   - En la ventana que nos aparece, buscamos el archivo que hemos transferido a nuestro ordenador, lo seleccionamos y pulsamos el botón “Abrir” como podemos ver en la Figura 2.27.

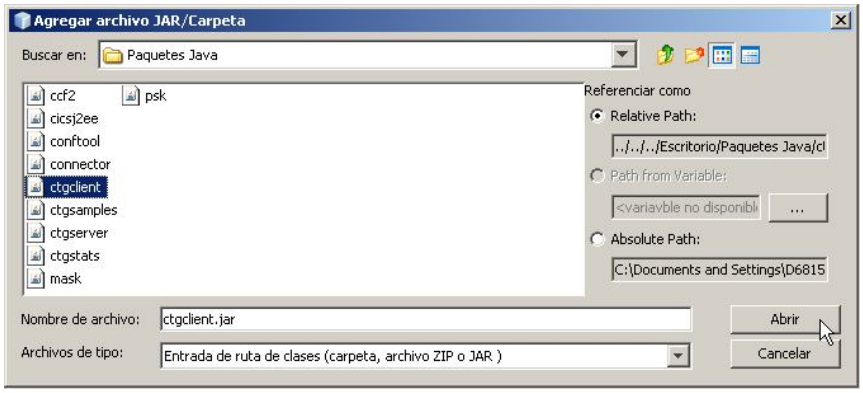


Figura 2.27 Agregar archivo ctgclient.jar

3. Al principio del archivo .java en el cual queramos hacer uso de las clases antes mencionadas, añadimos la siguiente instrucción
   - `import com.ibm.ctg.client.*;`

Con esto ya tendríamos todo preparado para hace uso de las clases necesarias para realizar peticiones al CTG.

Los pasos que vamos a seguir para realizar peticiones al CTG van a ser los siguientes:

1. Crear la conexión con el CTG
2. Crear la petición con los datos de entrada que vamos a enviar por la conexión
3. Enviar la petición y recoger la respuesta
4. Comprobar que no se a producido ningún error
5. Tratar los datos de salida

Por ejemplo, vamos a ver como invocar a un programa CICS llamado EJEMPLO1 al cual le pasas en una COMMAREA un DNI y te devuelve los datos que hay en el sistema de la persona con ese DNI. En un principio, el programa se haría más dinámico cogiendo la mayoría de los datos por medio de una interfaz gráfica en vez de, por ejemplo, poner la URL del CTG fija, para así poder realizar peticiones a más de un CTG, cambiar los datos de entrada, etc. Iremos comentando los diferentes pasos que vamos realizando y después del ejemplo explicaremos más en profundidad algunos de los pasos. Este ejemplo lo podemos ver en el Documento 2.5.

```
//Declaramos las variables de entrada que vamos a utilizar
JavaGateway jg;
ECIRequest er;
String URL;
int puerto;
String servidor;
String programa;
byte[] commarea;
int longitudCommarea;
String transid;
String dni;

//Inicializamos las variables con los datos que necesitamos
//URL donde se encuentra el CTG
URL = "20.20.20.1";
//Puerto donde se encuentra el CTG
puerto = 2006;
//Nombre del servidor CICS al que queremos hacer la petición ya que el CTG
//puede estar conectado a más de un CICS
servidor = "CICS1";
//Programa al que queremos hacer la petición
programa = "EJEMPLO1";
//Inicializamos el DNI y lo añadimos en forma de bytes a la COMMAREA que será
// la que le pasemos al programa
dni = "000000001";
commarea = dni.getBytes("ASCII");
//Longitud de la COMMAREA. Necesario para que funcione el programa
longitudCommarea = 281;


//Abrimos la conexión con el CTG
jg = new JavaGateway();
try {
```

```
	//Intentamos crear la conexión con el CTG en la URL y puerto indicados
	jg = new JavaGateway(URL, puerto);

} catch (IOException ioe) {
	//Error al abrir la conexión
	//Tratamos el error como corresponda, por ejemplo mostrando un mensaje
	//Comprobamos si la conexión sigue abierta y si es así la cerramos
	if (jg.isOpen() == true) {
		try { jg.close(); } catch (Throwable t2) { ; }
	}
	//Finalizamos la ejecución de la aplicación
	System.exit(1);

}catch (Exception e) {
	//Error de otro tipo al abrir la conexión
	//Realizamos los mismos pasos que hemos mencionado anteriormente
	if (jg.isOpen() == true) {
		try { jg.close(); } catch (Throwable t2) { ; }
	}
	System.exit(1);
}

//Creamos la petición
//Nombre de la transacción que queremos que se ejecute
//A continuación veremos la importancia de esto
transid = "F000";
//Inicializamos la petición con los parámetros adecuados
//Más adelante veremos con más detalle cuales son estos parámetros
//y los valores que pueden tomar
er = new ECIRequest(
	ECIRequest.ECI_SYNC,
	servidor,
	null,
	null,
	programa,
	transid,
	commarea,
	longitudCommarea,
	ECIRequest.ECI_NO_EXTEND,
	0);

//Lanzamos la petición
int iRc = 0;
try{
	//Lanzamos la petición por la conexión creada
	iRc = jg.flow(er);
} catch (IOException ioe) {
	//Error al invocar el programa
	//Lo tratamos como corresponda
	//Cerramos la conexión si sigue abierta y finalizamos la aplicación
	if (jg.isOpen() == true) {
		try { jg.close(); } catch (Throwable t2) { ; }
	}
	System.exit(1);
}

//Comrpobamos si se ha producido algún error
switch (er.getCicsRc()) {
	case ECIRequest.ECI_NO_ERROR:
		if (iRc == 0) {
			//No se han producido errores
```

```
                  //Salimos del case ya que no tiene sentido seguir
                  //comprobando errores
                  break;
            } else {
                  //Error en la Gateway
                  //Cerramos la conexión si sigue abierta y
                  //finalizamos la aplicación
                  if (jg.isOpen() == true) {
                        try { jg.close(); } catch (Throwable t2) { ; }
                  }
                  System.exit(1);
            }

      case ECIRequest.ECI_ERR_SECURITY_ERROR:
            //Error de seguridad
            //Cerramos la conexión si sigue abierta y
            //finalizamos la aplicación

            if (jg.isOpen() == true) {
                  try { jg.close(); } catch (Throwable t2) { ; }
            }
            System.exit(1);

      case ECIRequest.ECI_ERR_TRANSACTION_ABEND:
            //Error de ABEND (Abnormal End o final anormal)
            //al ejecutar el programa
            if (jg.isOpen() == true) {
                  try { jg.close(); } catch (Throwable t2) { ; }
            }
            System.exit(1);

      default:
            //Problema general invocando al programa
            //Cerramos la conexión si sigue abierta y
            //finalizamos la aplicación
            if (jg.isOpen() == true) {
                  try { jg.close(); } catch (Throwable t2) { ; }
            }
            System.exit(1);
}

//Cerramos la conexión si sigue abierta
if (jg.isOpen() == true) {
      try { jg.close(); } catch (Throwable t1) { ; }
}

//Recuperamos de la COMMAREA los datos que nos ha devuelto el programa
//que hemos invocado y los tratamos como corresponda
//En este caso sólo los mostramos por pantalla
String salida = new String(commarea, "ASCII");
```

Documento 2.5 Ejemplo de petición Java a programa de COMMAREA

A la hora de crear la conexión entre la aplicación y el CTG por medio de la clase JavaGateway, el formato del parámetro URL que le pasamos tiene que se <protocolo>://<dirección_ip_CTG>.

El protocolo puede ser TCP o SSL. Si no le indicamos nada coge por defecto TCP. Como en la mayoría de los casos usaremos TCP, cuando sea así no le indicamos nada.. Con esto la URL quedaría reducida a <dirección_ip_CTG>.

La dirección será la dirección IP donde estará escuchando el CTG que será la IP de la LPAR donde está instalado el CTG. Opcionalmente, si tenemos un servidor de DNS que

transforma las direcciones IP en nombres, le podemos pasar el nombre de la LPAR. Por ejemplo, en nuestro caso le podríamos pasar la IP 20.20.20.1 o el nombre PRUEBAS ya que tenemos un servidor DNS que se encarga de hacer la conversión entre ambos.

En general, el tratamiento de los errores será mostrar un mensaje al usuario indicándole el error que se ha producido. En algunos casos, como el error de seguridad, podríamos hacer un tratamiento diferente como podría ser solicitar un usuario o usuario y contraseña válidos.

Podríamos tratar cada uno de los errores que se pueden producir por separado pero hemos optado por tratar los más comunes como el error de seguridad o el error porque la transacción ha finalizado con un ABEND o finalización anormal. Agrupamos en el caso *default* el resto de errores que se pueden producir.

Aunque no debería suceder, si dejamos el programa de esta manera, al intentar crear la clase JavaGateway para abrir la conexión con el CTG el programa se queda parado y no pasa de esa instrucción. Para remediar esto, creamos una JavGateway en el main de la aplicación. El main, en el caso de usar NetBeans, se encontrará en el archivo <nombre_proyecto>App.java. Creamos la JavaGateway con cualquier URL y puerto. Esto nos producirá una excepción de Entrada/Salida pero la capturamos de la forma habitual y no hay ningún problema. De esta forma ya funciona correctamente la instrucción de crear la JavaGateway para abrir la conexión con el CTG.

Así quedaría el main de la aplicación para que todo funcione correctamente:

```
public static void main(String[] args) {

      JavaGateway jg;
      try {
            jg = new JavaGateway("localhost", 65000);
                  } catch (IOException ex) {
      }
      //A continuación está la única instrucción que había originalmente
      //en el main y que la dejamos como estaba
      launch(<nombre_proyecto>App.class, args);
}
```

Es especialmente importante la inicialización de la clase ECIRequest y de sus parámetros por lo que vamos a explicar más detalladamente cada uno de ellos y los valores que pueden tomar.

El constructor de la clase ECIRequest para peticiones con COMMAREA tiene los siguientes parámetros:

- Tipo de llamada
- Nombre del servidor CICS
- Usuario
- Contraseña
- Programa
- Transid
- COMMAREA
- Longitud de la COMMAREA
- Modo extendido
- Token LUW

Hay más constructores con diferentes parámetros, como no indicar el Modo extendido, un constructor para usar Canales y Contenedores, etc. pero este es el que vamos a usar para realizar peticiones a programas que usan COMMAREA.

Vamos a pasar a explicar para que sirve cada uno de estos parámetros, o campos de la clase, y los valores que pueden tomar.

2.4.1.1.- Tipo de llamada

Indica el tipo de llamada que queremos hacer con la petición. Para darle valor a este parámetro usaremos unas constantes definidas en la clase ECIRequest. Entre las diferentes llamadas que podemos hacer tenemos las siguientes que son las que más nos pueden interesar:

- Síncrona : Constante ECI_SYNC
- Síncrona con TPN : Constante ECI_SYNC_TPN
- Asíncrona : Constante ECI_ASYNC
- Asíncrona con TPN : Constante ECI_ASYNC

Las llamadas síncronas envían la petición y se quedan a la espera de recibir una respuesta. Por su parte, las asíncronas envían la petición y continúan ejecutando instrucciones. Tenemos que comprobar cada cierto tiempo que hemos recibido respuesta.

Los programas en CICS no tardan mucho en ejecutarse. Como mucho un par de segundos, normalmente medio segundo o menos. Esto, junto con el hecho de que si usamos servlets para desarrollar aplicaciones Web no nos permite hacer uso de los tipos ECI_ASYN y ECI_ASYN_TPN, hace que nos decantemos por usar peticiones síncronas.

Las llamadas sin TPN ejecutan el programa que le hayamos indicado bajo el control de la transacción por defecto del CTG que es la CSMI.

Las llamadas con TPN ejecutan también el programa que le hayamos indicado pero bajo el control de la transacción que le indiquemos en el parámetro Transid. Veremos más adelante como esto es muy importante por ejemplo para permitir que el programa pueda acceder a DB2.

Entre los tipos de llamadas, podemos encontrar útil el tipo CICS_EciListSystems. Esto nos devuelve los servidores CICS a los que podemos tener conectado el CTG. Nos devuelve los servidores conectados por IPIC definidos en las secciones IPICSERVER del archivo .ini de inicialización y los servidores conectados por EXCI definidos como DFHJVSYSTEM_XX en el miembro de variables de entorno. Esto sólo nos devuelve los servidores CICS que tenemos definidos, esto no quiere decir que realmente tengamos conexión con ellos.

Más fácil que usar esto, es usar la siguiente instrucción:

```
ECIRequest er = ECIRequest.listSystems(num_servidores);
```

Esto nos devuelve una petición ECI preparada para devolvernos, al lanzarla, los primeros num_servidores servidores definidos.

Hay que indicar que, entre los diferentes tipos de llamadas que hay, las conexiones IPIC no soportan los siguientes:

- ECI_GET_REPLY
- ECI_GET_REPLY_WAIT
- ECI_GET_SPECIFIC_REPLY

- ECI_GET_SPECIFIC_REPLY_WAIT
- ECI_STATE_ASYNC
- ECI_STATE_ASYNC_JAVA
- ECI_STATE_CANCEL
- ECI_STATE_CHANGED
- ECI_STATE_IMMEDIATE
- ECI_STATE_SYNC
- ECI_STATE_SYNC_JAVA

Esto nos limita los tipos de llamadas a las anteriormente mencionadas síncronas y asíncronas con o sin TPN y la CICS_EciListSystems.

2.4.1.2.- Nombre del servidor CICS

Este parámetro nos indica el nombre del servidor CICS al que queremos realizar la petición.

Puede tomar cualquier valor que indique un nombre de CICS definido en el archivo de inicialización .ini o en el miembro de variables de entorno.

Si indicamos un CICS no definido o uno que no está actualmente funcionando, la petición finalizará con el error ECI_ERR_NO_CICS.

2.4.1.3.- Usuario

Por medio de este parámetro le podemos indicar el identificador de usuario con el que queremos hacer la petición.

Esto nos sirve en caso de que tengamos configurada seguridad en el CTG.

Aunque los usuarios normalmente se definen en RACF con mayúsculas, podemos pasarle el usuario tanto en mayúsculas como en minúsculas ya que, normalmente, el sistema lo convierte automáticamente a mayúsculas.

Podemos introducir cualquier valor que indique un identificador de usuario que esté definido en RACF. En caso de no ser así, o de no introducir un identificador de usuario teniendo la seguridad del CTG habilitada, la petición finalizará con el mensaje de error ECI_ERR_SECURITY_ERROR.

Aunque no tengamos habilitada la seguridad, si le pasamos un identificador de usuario junto con un tipo de llamada con TPN, el programa se ejecuta bajo el identificador de usuario que le hayamos especificado. Esto puede ser útil por ejemplo por motivos de auditoría, para saber que usuarios ejecutan que programas y transacciones.

2.4.1.4.- Contraseña

Este parámetro nos va a permitir especificar una contraseña válida para el identificador de usuario que especifiquemos.

En algunos casos, como veremos más adelante, según como tengamos configurada la seguridad en el CTG puede que sea necesario indicar un identificador de usuario pero no una contraseña.

Normalmente las contraseñas de RACF contienen tanto mayúsculas como minúsculas por lo que el sistema no convierte todo automáticamente a mayúsculas como hace con el usuario. Esto hace que tengamos que respetar las mayúsculas y minúsculas al introducir la contraseña.

Si introducimos una contraseña errónea para el identificador de usuario introducido, o si no introducimos contraseña y es necesaria, la petición finalizará con el mensaje de error ECI_ERR_SECURITY_ERROR.

2.4.1.5.- Programa

Por medio de este parámetro indicaremos el identificador de programa CICS, de 8 caracteres máximo, que queremos que se ejecute.

El identificador del programa tiene que ser escrito en mayúsculas.

En caso de introducir un identificador de programa que no se encuentre instalado en el CICS, o si introducimos el nombre el minúsculas, la petición finalizará dando un error ECI_ERR_TRANSACTION_ABEND y nos indicará un código identificador de ABEND que normalmente será el AEI0 en caso de no encontrar el programa indicado.

2.4.1.6.- Transid

Este parámetro nos permite indicar un identificador de transacción CICS de 4 caracteres.

En caso de no indicarle un Transid, el programa que ejecutemos lo hará bajo el control de la transacción por defecto que es la CSMI.

Si indicamos un Trasnid y el tipo de llamada es ECI_SYNC_TPN, u otro con TPN, el programa se ejecutará bajo el control de la transacción que le indiquemos.

En caso de indicarle un Transid pero con el tipo de llamada ECI_SYNC, u otro sin TPN, el programa se ejecuta bajo el control de la transacción por defecto pero esta coge el alias que le indiquemos.

Las diferencias entre estas tres posibilidades y lo que ello implica lo explicaremos más adelante.

El identificador de transacción normalmente está definido en CICS en mayúsculas. Si le pasamos un identificador en minúsculas no se produce ningún error pero CICS no reconoce esta transacción y el resultado es como si no hubiésemos indicado ningún Transid.

2.4.1.7.- COMMAREA

En este parámetro le pasaremos el nombre de una variable de tipo byte[], es decir, un array de bytes. En esta variable introduciremos los datos de entrada que le tengamos que pasar al programa que vamos a ejecutar.

Cuando finalice la ejecución del programa, en esta variable encontraremos la respuesta que nos ha devuelto el programa tras su ejecución.

2.4.1.8.- Longitud de la COMMAREA

Este parámetro indicará la longitud de la COMMAREA que vamos a utilizar. Esto es muy importante ya que la longitud que indiquemos será la longitud de la COMMAREA que se envíe y reciba al hacer la petición.

Si este valor es mayor del necesario, estaremos enviando datos inútilmente con la sobrecarga de la red que esto conlleva.

Si este valor es menor que el necesario, no enviaremos o recibiremos todos los datos necesarios. Esto puede producir que no se envíen todos los datos de entrada necesarios para que el programa funcione o que recibamos menos datos de respuesta de los que deberíamos, perdiendo de esta manera información.

Normalmente, si indicamos un valor de longitud de COMMAREA menor que el que necesita el programa que queremos ejecutar para funcionar se producirá un ABEND. El código de este ABEND dependerá del programa.

Por otro lado, si el valor es mayor que el que necesita el programa, normalmente no se producirá ningún error aunque estaremos desperdiciando conexión. En algunos casos si que se producirá un ABEND y, al igual que antes, el código depende del programa.

2.4.1.9.- Modo extendido

Por medio de este parámetro vamos a poder manejar una LUW para realizar varias peticiones al CICS que podremos confirmar, *commit*, o deshacer, *backout* o *rollback*, de forma conjunta. También podemos realizar una petición simple que se confirma automáticamente al finalizar.

Como manejar LUWs por medio de este parámetro se explica más adelante.

Para realizar una petición simple de un programa y que se confirma le daremos el valor de la constante ECI_NO_EXTEND definida en la clase ECIRequest.

2.4.1.10.- Token LUW

Al igual que con el parámetro anterior, este parámetro nos permite manejar LUWs.

En caso de que queramos hacer una petición simple le daremos el valor de la constante ECI_LUW_NEW definida en la clase ECIRequest o directamente le daremos el valor 0.

2.4.3.- Transacción CICS ejecutada

Como hemos visto anteriormente, por medio del tipo de llamada y del parámetro Transid podemos controlar bajo que transacción se ejecuta el programa que hemos solicitado.

Si utilizamos un tipo de llamada con TPN, por ejemplo, ECI_SYNC_TPN, y le especificamos una transacción en Transid, el programa se ejecutará bajo la transacción indicada.

Si no utilizamos un tipo de llamada con TPN, por ejemplo ECI_SYNC, pero especificamos en Transid una transacción válida, el programa se ejecuta bajo la transacción por defecto aunque a esta se le da el alias especificado en Transid. La transacción por defecto es la CSMI para el CTG de z/OS y la CPMI para el CTG multiplataforma.

Por último, si no usamos una llamada con TPN ni especificamos un Transid, el programa se ejecuta bajo la transacción por defecto.

La transacción por defecto, la CSMI en nuestro caso, no tiene permisos de acceso a DB2, a no ser que se los hayamos dado. Si hacemos una petición de un programa que acceda a DB2, como el EJEMPLO1 usado anteriormente, sin usar uno de los dos primeros métodos, no se produce un error al ejecutar el programa pero en vez de devolvernos los datos que debería nos da un error SQL -923. Este error se da, entre otras razones, cuando tenemos un acceso restringido a DB2 como es nuestro caso.

Si queremos hacer uso de un programa que acceda a DB2 tenemos que ejecutarlo, o bien bajo una transacción con acceso a DB2 o bien bajo la transacción por defecto pero dándole el alias de una transacción con acceso a DB2.

Hay que tener en cuenta que la transacción bajo la que se ejecuta el programa, la CSMI si no usamos una llamada con TPN o la indicada en el parámetro Transid en caso contrario, tiene que apuntar obligatoriamente al programa DFHMIRS que es el que se arranca primeramente cuando hacemos uso del CTG.

No vamos a explicar como hacer que una transacción apunte a un programa determinado o como darle acceso a DB2 por considerar que queda fuera del ámbito de este proyecto.

Cuando ejecutamos una transacción con el alias de otra, lo que hacemos es darle a la primera las características de la segunda. Entre otras cosas, esto puede ser útil para:

- Control de recursos y comandos
- Asignación de prioridades iniciales
- Asignación de selección de plan de base de datos, lo que significa acceso a DB2

La mayor diferencia entre ejecutar una transacción con el alias de otra, o ejecutar directamente la segunda transacción, es que con el segundo método podemos comprobar más fácilmente las transacciones ejecutadas, lo que nos puede ser útil para motivos de auditoría.

2.4.4.- Manejo de LUWs

Como hemos indicado anteriormente, una LUW es un grupo de transacciones o programas, a los cuales hacemos *commit* o *rollback* en grupo, como si fuesen uno solo. Esto quiere decir que si hacemos *commit* lo hacemos sobre los cambios que han realizado todos los programas y, si uno falla y hace *rollback*, hacemos *rollback* sobre los cambios que han realizado todos los programas.

Una LUW finaliza cuando se dan uno de los siguientes sucesos:

- Hacemos *commit* o *rollback* específicamente sobre la LUW
- Una de los programas ejecuta de forma específica un *commit*
- Se produce un error y se ejecuta un *rollback* automáticamente

Para manejar las LUWs vamos a utilizar principalmente tres atributos de las peticiones ECI que son:

- Extend_Mode: modo de la petición ECI
- Program: nombre del programa que queremos que se ejecute en el CICS
- Luw_Token: token que identifica a la LUW que estamos manejando

En la Tabla 2.4 podemos ver una relación entre estos atributos y los valores que les tenemos que dar dependiendo de la operación que queramos realizar sobre una LUW.

Tarea a realizar	**Valor de los atributos**
Llamada a un programa que es el único de una LUW. *Se lanza una petición del cliente al servidor y una respuesta es enviada al cliente sólo cuando todos los cambios realizados por el programa han sido confirmados.*	Extend_Mode = ECI_NO_EXTEND
	Program = *especificar*
	Luw_Token = 0 o ECI_LUW_NEW
Llamada a un programa que es el primero de una LUW.	Extend_Mode = ECI_EXTENDED
	Program = *especificar*
	Luw_Token = 0 o ECI_LUW_NEW
	Guardar el valor del Luw_Token de la respuesta
Llamada a un programa que es la continuación de una LUW.	Extend_Mode = ECI_EXTENDED
	Program = *especificar*
	Luw_Token = *valor_guardado*
Llamada a un programa que es la última	Extend_Mode = ECI_NO_EXTEND

de una LUW y confirmar los cambios.	Program = *especificar*
	Luw_Token = *valor_guardado*
Finalizar una LUW existente y confirmar los cambios.	Extend_Mode = ECI_COMMIT
	Program = NULL
	Luw_Token = *valor_guardado*
Finalizar una LUW existente y desechar los cambios.	Extend_Mode = ECI_BACKOUT
	Program = NULL
	Luw_Token = *valor_guardado*

Tabla 2.4 Atributos y valores para manejar LUWs

Si estamos utilizando conexiones EXCI, podemos habilitar una traza llamada JNI donde podemos observar información sobre las diferentes peticiones a una LUW.

Para habilitar la traza JNI, en el miembro de variables de entorno añadimos las siguientes líneas:

- CTG_JNI_TRACE_ON=YES
- CTG_JNI_TRACE=/SYSTEM/tmp/ctg1_jni_trace.txt

La primera línea habilita la traza JNI y la segunda indica la ruta del fichero en la parte Unix donde queremos que se almacene la traza.

Si tenemos arrancado el CTG y añadimos estas líneas al miembro de variables de entorno, tendremos que parar y volver a arrancar el CTG para que coja los cambios introducidos.

Una vez que tenemos habilitada la traza JNI, ejecutamos una LUW haciendo dos peticiones a un programa llamado EC01 y por último haciendo *commit* sobre la LUW. En el archivo de la traza JNI, entre otras cosas, podemos ver lo siguiente:

Primera petición

```
CTG6800T ECI parameters on entry: Call_Type=1, Extend_Mode=1, Luw_Token=0
CTG6805T ECI parameters on exit:  Call_Type=1, Extend_Mode=1,
Luw_Token=16777216, Commarea_Length=53, Cics_Rc=0, AV=0
```

Segunda petición

```
CTG6800T ECI parameters on entry: Call_Type=1, Extend_Mode=1,
Luw_Token=16777216
CTG6805T ECI parameters on exit:  Call_Type=1, Extend_Mode=1,
Luw_Token=16777216, Commarea_Length=53, Cics_Rc=0, AV=0
```

Petición de commit

```
CTG6800T ECI parameters on entry: Call_Type=1, Extend_Mode=2,
Luw_Token=16777216
CTG6805T ECI parameters on exit:  Call_Type=1, Extend_Mode=2, Luw_Token=0,
Commarea_Length=0, Cics_Rc=0, AV=0
```

Observamos como el valor del token LUW en la entrada de la primera llamada vale 0, correspondiente a una nueva LUW, mientras que en la salida toma un valor de 16777216. Este valor será el que tengamos que almacenar e indicar en sucesivas peticiones que queramos que pertenezcan a la misma LUW.

También podemos observar como el valor de Modo Extendido en las dos primeras llamadas vale 1 y en la tercera, donde hemos puesto el Modo Extendido a ECI_COMMIT, vale

2. Estos valores corresponden a los valores de las constantes ECI_EXTENDED y ECI_COMMIT que hemos utilizado.

Si estamos utilizando IPIC, para ver si se ha ejecutado la LUW podemos hacer uso de las estadísticas que muestra el CTG. Aunque más adelante veremos en más profundidad las estadísticas, ahora nos basta con fijarnos en las estadísticas GD_LLUWTXNC, GD_ILUWTXNC, GD_LLUWTXNR y GD_ILUWTXNR. Las dos primeras nos dicen las LUWs confirmadas desde que arranco el CTG y en el intervalo actual mientras que las dos últimas nos dicen las LUWs a las que se ha hecho *rollback* desde que arranco el CTG y en el periodo actual.

Si comprobamos estas estadísticas antes y después de ejecutar una LUW veremos que estos valores cambian. En nuestro caso, a aumentado el valor en 1 de las estadísticas GD_LLUWTXNC y GD_ILUWTXNC.

2.4.5.- Conversión de datos

Los mainframes utilizan un código de caracteres de 8 bits llamado EBCDIC mientras que un ordenador personal usa el código ASCII. Por ello tenemos que hacer una conversión de los datos que vamos a transmitir entre la aplicación cliente y el mainframe. Esto sólo hay que realizarlo si usamos programas basados en COMMAREA, no hace falta realizarlo si usamos Canales y Contenedores de tipo CHAR. Esta conversión la podemos hacer en el mainframe o en la aplicación cliente.

2.4.5.1.- Conversión en el mainframe

Para que la conversión de los datos se realice en el mainframe, tendremos que hacer uso de la tabla de conversión de páginas de códigos llamada DFHCNV.

Para cada programa en el CICS que queramos usar, tendremos que introducir en la DFHCNV ese programa para que los datos sean traducidos de EBCDIC a ASCII y viceversa.

De esta forma en nuestro programa lo único que tendremos que hacer es coger el array de bytes que nos devuelva el CTG simulando la COMMAREA y convertirlo en una cadena de texto para poder manejarlos o convertir los datos de entrada en un array de bytes para mandárselo al CTG, sin cambiar la página de códigos.

En la Figura 2.28 podemos ver una muestra de la tabla DFHCNV donde convierte los datos de los programas EC01 y EC02, programas de ejemplo, de la página de códigos IBM037, página de códigos en inglés del mainframe, a la página de códigos 8859-1 o ISO Latin-1.

```
*---------------------------------------------------------------
         DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=EC01,USREXIT=NO,             *
               SRVERCP=037,CLINTCP=8859-1
         DFHCNV TYPE=SELECT,OPTION=DEFAULT
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767,   *
               LAST=YES
*---------------------------------------------------------------
         DFHCNV TYPE=ENTRY,RTYPE=PC,RNAME=EC02,USREXIT=NO,             *
               SRVERCP=037,CLINTCP=8859-1
         DFHCNV TYPE=SELECT,OPTION=DEFAULT
         DFHCNV TYPE=FIELD,OFFSET=0,DATATYP=CHARACTER,DATALEN=32767,   *
               LAST=YES
```

Figura 2.28 Muestra de la tabla DFHCNV

2.4.5.2.- Conversión en la aplicación cliente

La otra opción que tenemos para convertir los datos que vamos a manejar es hacerlo desde la aplicación cliente. Para esto, en cada uno de las aplicaciones cliente que vayan a acceder al CTG tendremos que añadir líneas extra de programación para convertir los datos que intercambiemos con el CTG. En el Documento 2.6 podemos ver como hacer esto desde Java mientras que cuando expliquemos como desarrollar aplicaciones en .Net lo explicaremos para este lenguaje.

```
byte[] COMMAREA = entrada.getBytes(String codificación);
String resultado = new String(byte[] COMMAREA, String codificación);
```

Documento 2.6 Conversión de datos a intercambiar con el mainframe

La primera instrucción convierte la cadena de texto entrada, donde tenemos los datos que queremos pasar al programa CICS que queremos ejecutar, a un array de bytes que pasaremos al CTG simulando la COMMAREA. Este array de bytes se codifica con la página de códigos que le indiquemos que para que sea la IBM037, utilizada por el CTG y el CICS, le pasaremos la cadena de texto 037 o IBM037.

La segunda instrucción hace el proceso contrario. Coge el array de bytes que le devuelve el programa CICS y lo convierte a una cadena de texto haciendo uso de la página de códigos que le indiquemos en el segundo argumento para decodificar el array de bytes. Para obtener los datos correctos le indicaremos que la página de códigos es la IBM037.

Algunos tipos de datos no se convierten correctamente haciendo uso del segundo método. Esto ocurre por ejemplo con los tipos de datos CVDA. Este tipo de datos son constantes numéricas definidas en el mainframe que se corresponden con valores entendibles por el usuario. Por ejemplo, los valores 147, 148 y 149 indican lenguajes de programación compatibles con el CICS como Java, C++ o C respectivamente. Este tipo de datos se utiliza en algunos comandos o para almacenar cierta información.

Si utilizamos programas que devuelvan datos de este tipo tendremos que usar el primer método de conversión o convertir estos datos en el programa CICS a caracteres antes de devolverlos o no recibiremos los datos correctos.

2.4.2.- Aplicaciones .Net

Otro de los lenguajes de programación desde los que podemos hacer peticiones al CTG es desde .Net, ya sea VisualBasic.Net o C#.Net. Aunque ambos lenguajes de programación son diferentes, los dos usan la misma API para establecer conexión con el CTG y realizar peticiones.

A la hora de hacer peticiones desde aplicaciones .Net principalmente usaremos las clases GatewayConnection y EciRequest similares a las clases JavaGateway y ECIRequest de Java.

Para poder hacer uso de estas clases deberemos importar en nuestro proyecto la librería IBM.CTG.Client.dll y que la librería ctgclient.dll se encuentre en el Path del sistema. La primera se encuentra dentro del *SupportPac ca 73* que podemos descargar desde la página oficial de IBM mientras que la segunda se puede solicitar a IBM al adquirir la licencia de CTG.

Para importar la librería a nuestro proyecto en el Visual Studio 2005 que vamos a utilizar para desarrollar aplicaciones .Net seguiremos los siguientes pasos:

1. Una vez creado el proyecto, en el “Explorador de soluciones” hacemos clic derecho sobre el nombre del proyecto y seleccionamos “Agregar referencia…” como podemos ver en la Figura 2.29.

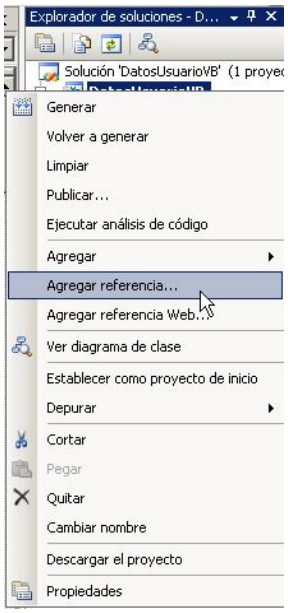


Figura 2.29 Agregar referencia

2. Nos aparecerá la ventana que podemos ver en la imagen 2.30 donde seleccionamos la pestaña "Examinar" y ahí buscamos donde tenemos la librería IBM.CTG.Client.dll, la seleccionamos y pulsamos aceptar.

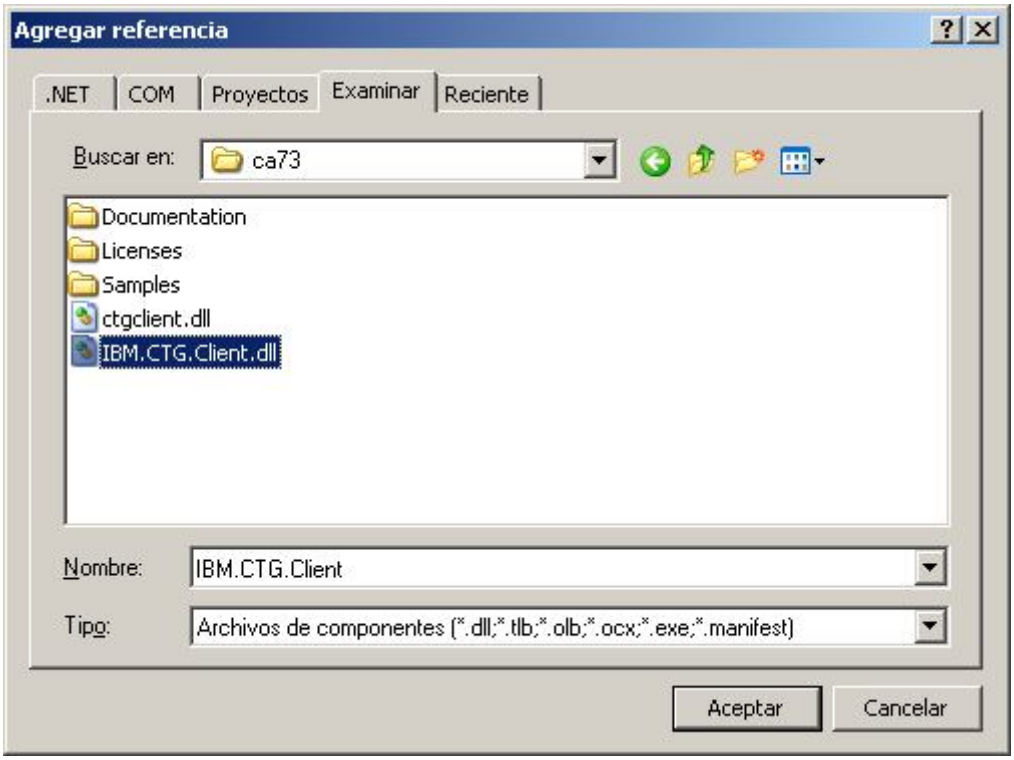


Figura 2.30 Ventana de selección de la dll

Una vez hecho esto ya podemos hacer uso de las clases que contiene. Los pasos que seguiremos para realizar peticiones al CTG serán los mismos que seguimos en las aplicaciones Java.

En el Documento 2.7 podemos ver un ejemplo de una aplicación escrita en VisualBasic.Net para realizar peticiones al programa EJEMPLO1 que hemos visto anteriormente. En este caso, los comentarios van precedidos de ' .

```
'Importamos el paquete que contiene las clases GatewayConnection y EciRequest
Imports IBM.CTG

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
'Definimos las variables que vamos a necesitar
        Dim jg As GatewayConnection
        Dim er As EciRequest
        Dim URL As String
        Dim puerto As Integer
        Dim servidor As String
        Dim programa As String
        Dim longitudCommarea As Integer
        Dim transid As String

'Inicializamos las variables
        URL = "20.20.20.1"
        puerto = 2006

        servidor = "CICS1"
        programa = "EJEMPLO1"
        transid = "F000"

        Dim dni As String
        dni = "000000001"
        longitudCommarea = 281

'Creamos la conexion con el CTG
        Try
            jg = New GatewayConnection(URL, puerto)
        Catch ex As Exception
            'Error al crear la conexión
            Dim st1 As String = "Error al crear la conexión con el CTG" +
vbCrLf + ex.Message
            MessageBox.Show(st1)
            End
        End Try

'Creamos la Peticion
        er = New EciRequest()
        er.ServerName = servidor
        er.UserId = usuario
        er.Program = programa
        er.ExtendMode = EciExtendMode.EciNoExtend
       Try
'Pasamos los datos de entrada a la COMMAREA convirtiéndolos a la página de
'códigos 037
            er.SetCommareaData(System.Text.Encoding.GetEncoding("037").GetByt
       es(dni)
        Catch ex As Exception
            Dim st1 As String = "Error al codificar los datos de entrada" +
vbCrLf + ex.Message
            MessageBox.Show(st1)
            End
        End Try
        er.TransId = transid

'Lanzamos la petición
        Dim iRc As Integer = 0
```

```
        Try
            iRc = jg.Flow(er)
        Catch ex As Exception
            Dim st1 As String = "Error al lanzar la petición" + vbCrLf +
ex.Message
            MessageBox.Show(st1)
            End
        End Try

'Comprobamos si se han producido errores
        Dim st As String
        Dim err As Integer = 0
        Select Case er.ReturnCode
            Case EciReturnCode.EciNoError
                If iRc.Equals(0) Then
                    'Ejecución correcta de la petición
                Else
                    'Error en la conexión al lanzar el programa
                    MessageBox.Show("Error en la Gateway" +
er.ReturnCodeString)
                    err = 1
                End If
            Case EciReturnCode.EciErrSecurityError
                'Error de seguridad
                MessageBox.Show("Error de seguridad")
                err = 1
            Case EciReturnCode.EciErrTransactionAbendç
                'Error de ABEND
                st = "The transaction abended: " + er.AbendCode
                MessageBox.Show(st)
                err = 1
            Case EciReturnCode.EciErrNoCics
                'Error al no encontrar el servidor CICS especificado
                MessageBox.Show("No se encuentra el CICS indicado")
                err = 1
            Case Else
                'Otro tipo de error
                Dim st1 As String = ("ECI return code: " +
er.EciReturnCode.ToString() + " (" + CType(er.EciReturnCode, Integer) + ")")
                MessageBox.Show(st1)
                err = 1
        End Select

'Cerramos la conexión si sigue abierta
        If (jg.IsOpen) Then
            jg.Close()
        End If

'Recogemos los datos y se los mostramos al usuario
        Dim resultado As String
        resultado =
      System.Text.Encoding.GetEncoding("ASCII").GetString(er.GetCommarea())

        MessageBox.show(resultado)
    End Sub
End Class
```

Documento 2.7 Ejemplo de petición VB.Net a programa de COMMAREA

Vemos como la estructura del programa es similar a la del programa Java excepto que para dar a cada atributo de la clase EciRequest su valor adecuado, en vez de hacerlo por medio de un constructor lo hacemos individualmente.

En este caso no podemos indicarle tipo de llamada. Todas son síncronas.

Para controlar la transacción que se ejecuta y el alias hacemos uso de los atributos EciRequest.TransId y EciRequest.MirrorTransId. Con el primero ejecutamos la transacción por defecto con el alias especificado en el atributo mientras que con el segundo ejecutamos directamente la transacción especificada en el atributo.

Vemos como la forma de convertir los datos también ha cambiado.

Si queremos convertir los datos en la aplicación hacemos uso de las instrucciones que podemos ver en el Documento 2.8.

```
er.SetCommareaData(System.Text.Encoding.GetEncoding("037").GetBytes(dni)
System.Text.Encoding.GetEncoding("037").GetString(er.GetCommarea())
```

Documento 2.8 Conversión de datos a intercambiar

Con la primera instrucción convertimos los datos de la cadena dni a la página de códigos IBM037 y lo pasamos a un array de bytes.

Con la segunda instrucción hacemos el proceso contrario.

2.5.- Prueba con Canales y Contenedores

A la hora de hacer peticiones a programas CICS podemos utilizar programas que funcionen por medio de COMMAREA o por medio de Canales y Contenedores para intercambiar datos.

Este segundo método presenta varias ventajas con respecto al uso de la COMMAREA siendo las que más nos afectan a nosotros las siguientes:

- Intercambio de cantidades de datos superiores a los 32KB limitados por la COMMAREA
- Posibilidad de estructurar los datos
- Envío de datos únicamente modificados o nuevos, no de todos los datos de un Canal

Para poder utilizar programas CICS que funcionen con Canales y Contenedores tenemos que hacer uso de la nueva conexión IPIC aparecida en la versión 3.1 del CICS.

Si tenemos el CTG v7.2 estamos limitados a usar aplicaciones de cliente escritas en Java para hacer uso de este tipo de programas. Si usamos el CTG v8.0 también podemos hacer peticiones a programas CICS que usen Canales y Contenedores desde aplicaciones escritas en .Net.

En el Documento 2.8 podemos ver un ejemplo de programa escrito en Java que hace una petición al programa ECICHAN y un método para pasar datos de bytes a entero. El programa ECICHAN es un ejemplo para probar programas CICS que usa Canales y Contenedores. Simplemente recibe unos datos de entrada y devuelve los mismos datos junto con su longitud, la hora actual del CICS y algún dato más.

Los Contenedores pueden ser de tipo CHAR, o texto normal, o de tipo BIT, que sirven para enviar texto pero también números, etc. Veremos la forma tanto de crear como de recibir ambos tipos de Contenedores.

```
try {
```

```
	//Definimos las variables que vamos a necesitar
	String URL;
	int puerto;
	String servidor;
	String programa;
	String nombreCanal;
	String nombreContenedorEntrada;
	String nombreContenedorError;
	String datos;

	//Las inicializamos con los valores adecuados
	//Normalmente el nombre que le demos al Canal no importa ya que los
	//programas CICS trabajan con el Canal que les llega sin comprobar
	//su nombre
	//El nombre del Contenedor SI es importante
	URL = "20.20.20.1";
	puerto = 2006;
	servidor = "CICS1";
	programa = "ECICHAN";
	nombreCanal = "CTGCHANNEL";
	nombreContenedorEntrada = "INPUTDATA";
	nombreContenedorError = "ERRORDARA";
	datos = "Hola CICS";

	//Abrimos la conexión con el CTG
	JavaGateway javaGateObject = new JavaGateway(URL, puerto);

	//Creamos el Canal
	Channel theChannel = new Channel(nombreCanal);

	//Creamos un contenedor con el nombre que espera que tenga el programa
	//CICS. Este nombre lo tenemos que saber de antemano.
	//De esta forma creamos un Contenedor de tipo CHAR
	theChannel.createContainer(nombreContenedorEntrada, datos);
	//De esta forma crearíamos un Contenedor de tipo BIT en caso necesario
	//theChannel.createContainer(nombreContenedorEntrada,
datos.getBytes("IBM-037"));


	//Creamos la petición que vamos a enviar
	ECIRequest er = new ECIRequest(
		ECIRequest.ECI_SYNC,
		servidor,
		null,
		null,
		programa,
		null,
		theChannel,
		ECIRequest.ECI_NO_EXTEND,
		0);

	//Lanzamos la petición por la Conexión
	javaGateObject.flow(er);


	if (er.getRc() == 0) {  //Comprobamos que no se han producido errores

		try {
		//Comprobamos si hay un contenedor llamado ERRORCONT ya que
		//sabemos que el programa deja los errores que se han producido
		//en este Contenedor
```

```
      Container errorcont =
theChannel.getContainer(nombreContenedorError);

} catch (ContainerNotFoundException e) {
//Si se produce una excepción porque no se encuentra el
//Contenedor ERRORCONT suponemos que no se han producido errores

} catch (Exception e) {
//Capturamos cualquier otra excepción que se pueda producir
//al recuperar el Contenedor de errores
}

//Como no sabemos los nombre de los Contenedores donde el
//programa CICS nos va a devolver los datos, recorremos todos
//los Contenedores que hay en el Canal tratando sus datos
Iterator<Container> it = theChannel.getContainers().iterator();
while (it.hasNext()) {

      //Recogemos el siguiente Contenedor del Canal
      Container cont = it.next();
      //Como no sabemos el tipo de Contenedor, comprobamos de que
      //tipo es y lo tratamos como corresponda según su tipo
      switch (cont.getType()) {
            //Contenedor de tipo CHAR (cadena de texto)
            case CHAR:
                  //Recuperamos los datos del Contenedor
                  //No hay que hacer ninguna transformación
                  String dataStr = cont.getCHARData();
                  //Mostramos los datos junto al nombre del
                  //Contenedor
                  System.out.print("\t[CHAR]-"
                        +cont.getCCSID()
                        + "\t" + cont.getName()
                        + "("
                        + dataStr.length() + " bytes)"
                        + " = ");

                  System.out.println(dataStr);
                  break;

            //Contenedor de tipo BIT
            case BIT:
                  //Recuperamos los datos del Contenedor a un
                  //array de bytes
                  byte[] data = cont.getBITData();
                  //Mostramos el nomrbre del Contenedor y
                  //su longitud
                  System.out.print("\t [BIT] "
                        + "\t\t"
                        + cont.getName()
                        + "("
                        + data.length
                        + " bytes)"
                        + " = " );

                  //Pasamos el array de bytes a texto
                  //convirtiéndolo con la página de códigos 037
                  String str2 = new String(data, "IBM037");
                  System.out.print("String: " + str2 + " ");

                  //Lo pasamos a entero con el método que hay
```

```
                              //más abajo
                              int sum = bytesToInt(data, data.length);
                              System.out.print("Decimal:");
                              System.out.print(((Integer) sum).toString() + "
                        ");

                              //Lo pasamos a hexadecimal
                              System.out.print("  X'");
                              for (int i = 0; i < data.length; i++) {
                                    String hex = "0" +
                                    Integer.toHexString(data[i]);
                                    System.out.print(hex.substring(
            hex.length()-2));
                              }
                              System.out.println("'");
                        }
                  }
            } else {
            //Se ha producido un error
            //Comprobamos los diferentes tipos de errores, de seguridad, ABEND,
            //etc. como hacíamos en las peticiones de COMMAREA
            }

}catch (IOException ioe) {
      //Error al crear la conexión con el CTG
      System.out.println("No se puede conectar con el CTG:"+ ioe.toString());
}catch (Exception e) {
      //Otro tipo de errores
      System.out.println("Excepcion ocurrida: " + e.toString());
}

//Cerramos la conexión si sigue abierta
if (javaGateObject.isOpen() == true) {
      try { javaGateObject.close(); } catch (Throwable t2) { ; }
}

//Método para pasar los datos de hexadecimal a entero
public static int bytesToInt(byte[] bytesToConvert, int length) {
      int result = 0;

      for (int i = 0; i < length; i++) {
            result |= ((int) bytesToConvert[i] & 0x000000ff) << 8 * (length -
1 - i);
      }
      return result;
}
```

Documento 2.8 Ejemplo Java para llamar a programas de Canales y Contenedores.

Podemos comprobar como el primer cambio que se produce entre este código y el de llamadas a programas de COMMAREA es el constructor de la clase ECIRequest. Ahora en vez de pasarle un array de bytes que simula la COMMAREA le pasamos una variable de la clase Channel con los Contenedores de entrada que corresponda. En esta variable nos devolverá el programa CICS los Contenedores de salida. Vemos como a la hora de crear y recibir un Contenedor de tipo CHAR no hace falta hacer ninguna conversión entre códigos. Esta es otra ventaja entre usar Canales y Contenedores a usar COMMAREA.

El segundo cambio es a la hora de tratar los datos de salida. En el caso de la COMMAREA no sabíamos donde acababa un dato y empezada el siguiente a no ser que supiésemos de antemano su longitud. Ahora normalmente tenemos cada dato separado en un

Contenedor de forma que lo podemos separar tratar más cómodamente. En el caso de no saber el nombre de los Contenedores que nos va a devolver el programa CICS, como en el ejemplo, podemos recorrer todos los Contenedores comprobado su tipo y tratando sus datos. En caso de conocer los nombres y tipos de Contenedores que nos van a llegar, podemos directamente recoger los Contenedores que nos interesan y tratar sus datos como corresponda. Para recoger un Contenedor específico usamos la instrucción
Container c = <nombreVariableChannel>.getContainer(<nombreContenedor>);

Si entre los datos que vamos a enviar o recibir por medio del CTG queremos enviar o recibir el carácter Ñ o ñ vamos a tener que usar la página de códigos IBM01145 en vez de la IBM037.

2.6.- Seguridad

Como ya hemos comentado, es posible hacer que el CTG compruebe cuando le llegan peticiones si el usuario o el usuario y contraseña que realiza la petición está en RACF. Para que esto suceda tenemos que habilitar la seguridad del CTG.

Dependiendo de si usamos conexiones EXCI o IPIC el modo de habilitar esta seguridad cambia. A continuación vamos a ver como habilitar la seguridad en conexiones IPIC. Nos centramos en las conexiones IPIC porque, como ya hemos dicho, parece que IBM tiende a dar más importancia a IPIC además de ser la única manera de poder usar Canales y Contenedores.

Para habilitar la seguridad en conexiones IPIC tenemos que utilizar uno de los parámetros del recurso IPCONN. Tal y como tenemos realizada la instalación del CTG, el recurso IPCONN se instala automáticamente cada vez que se necesita, con unos parámetros por defecto.

Para evitar que esto suceda, tenemos que modificar el parámetro Urm en la definición del recurso TCPIPSERVICE que hemos definido anteriormente. Para ello, después de identificarnos en el sistema con la transacción CESN, ejecutamos el comando CEDA ALTER TCPIPSERVICE(CTG1A) G(IPCTG1A). De esta forma nos aparecerá una pantalla similar a la de la Figura 2.20 que hemos visto en la configuración de la conexión EXCI. Modificamos el parámetro Urm y le damos el valor “NO”. Pulsamos Control para que se verifiquen los cambios. De esta forma ya no se va a realizar una instalación automática de IPCONN.

Una vez que ya hemos deshabilitado la instalación automática del IPCONN vamos a definir nuestro propio recurso IPCONN que nos permita habilitar la seguridad del CTG. Para definir un recurso IPCONN de nombre CTG1A en el grupo IPCTG1A ejecutamos el comando CEDA DEF IPCONN(CTG1A) G(IPCTG1A) y le damos los siguientes valores además de los que aparecen por defecto:

- Ipconn : CTG1A
- Group : IPCTG1A
- APplid : CTG1A
  - El nombre del CTG al que queremos que se conecte
- Networkid : VTAM
- Host : 20.20.20.1
  - IP donde está el CICS
- Tcpipservice : CTG1A
  - Nombre que le hemos dado al TCPIPSERVICE asociado
- Receivecount : 100
  - Número de clientes que se pueden conectar al CTG
- Userauth : Identify

- Tipo de seguridad

El parámetro que más nos interesa es Userauth ya que es el que nos especifica el tipo de seguridad que vamos a tener en el CTG. Los diferentes valores que puede tomar este parámetro y la funcionalidad que nos ofrecen son:

- Local
  - No hay que pasarle usuario ni contraseña aunque se puede hacer
  - Las peticiones corren bajo el usuario que se le pasa o bajo el usuario por defecto, DFLTUSER, si no se le pasa ninguno
- Identify
  - Hay que pasarle un usuario válido de RACF
  - Las peticiones corren bajo este usuario
- Verify
  - Hay que pasarle un usuario y una contraseña válidos de RACF
  - Las peticiones corren bajo este usuario
- Defaultuser
  - No se le puede pasar usuario ni contraseña
  - Las peticiones corren bajo el usuario por defecto

Para habilitar seguridad en el CTG le tenemos que dar un valor Identify o Verify. Para pasar usuario y contraseña usamos los atributos correspondientes de la clase ECIRequest como hemos visto anteriormente.

Hemos probado ambas opciones de seguridad, Identify y Verify, y funcionan correctamente. En caso de no pasarle un usuario o un usuario y contraseña inválidos, dependiendo de la opción elegida, el CTG nos devuelve un error ECI_ERR_SECURITY_ERROR aunque no se especifica si el error se ha producido en el usuario o en la contraseña.

2.7.- SSL

En vez de utilizar sólo conexiones TCP entre la aplicación cliente y el CTG podemos utilizar conexiones TCP con SSL. Esto permite introducir un factor de autenticidad y privacidad en nuestras comunicaciones ya que permite comprobar que el mainframe donde está instalado el CTG es quien dice ser y encripta las conexiones entre la aplicación y el CTG.

Para poder hacer uso de SSL, la conexión entre el CTG y el CICS puede ser tanto EXCI como IPIC.

2.7.1.- Configuración de SSL

El objetivo es crear un certificado auto-firmado que sirva para autentificar a quien lo ha generado y que además contenga una clave pública y privada que permita a quien reciba datos de él desencriptarlos.

Para generar este certificado haremos uso de una herramienta llamada keytool que forma parte del SDK de Java. Esta herramienta permite para crear un almacén de certificados llamado key ring o keystore, y generar un par de claves, pública y privada, para una máquina. Este par de claves se guardan en un certificado auto-firmado.

El uso de la herramienta keytool se hace desde la parte Unix del mainframe y desde Windows por medio de una consola de comandos por medio del comando del mismo nombre.

Cuando utilicemos el comando keytool por primera vez en la parte Unix del mainframe nos puede aparecer el error que se puede observar en la Figura 2.31. Esto ocurre porque la memoria que el usuario puede usar en su sesión de TSO es insuficiente para hacer uso de esta herramienta.

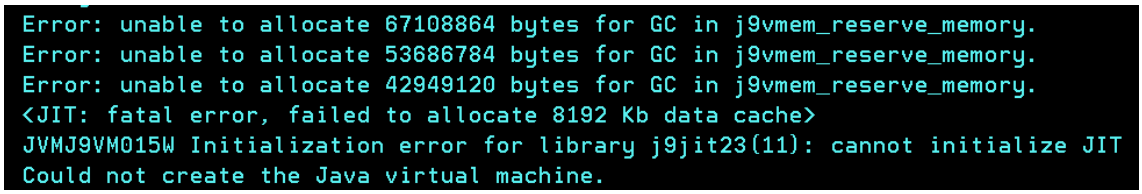


Figura 2.31 Error por memoria insuficiente

Si nos ocurre este error, tenemos que aumentar la cantidad de memoria disponible en nuestra sesión de TSO. Para ello, en la pantalla de Logon de TSO, que podemos ver en la Figura 2.32, tenemos que modificar el parámetro Size a un valor adecuado. Un valor de 123000 es suficiente.

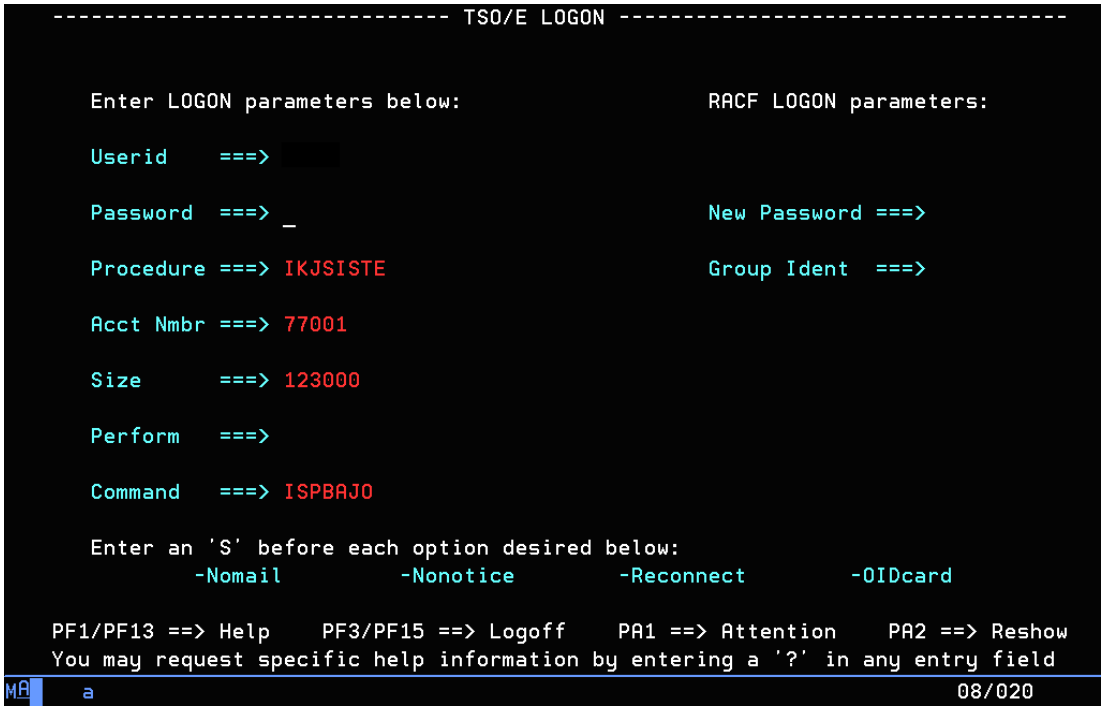


Figura 2.32 Pantalla de Logon de TSO con parámetro Size modificado

Esta modificación se almacena por lo que no tenemos que volver a realizarla cada vez que iniciemos una sesión de TSO y vayamos a utilizar el comando keytool.

Para hacer uso del comando keytool en el mainframe tenemos que acceder a la parte Unix. Como ya hemos visto, esto lo podemos hacer por medio de OMVS o de ISH. Si accedemos por medio de OMVS, la longitud de la línea de comandos es insuficiente para la longitud de los comandos que vamos a ejecutar. Por eso, utilizaremos la línea de comandos de ISH. Para acceder a ISH, desde cualquier panel de TSO tecleamos el comando tso ish.

Una vez hayamos teclado el comando nos aparecerá la pantalla que podemos ver en la figura 2.33.

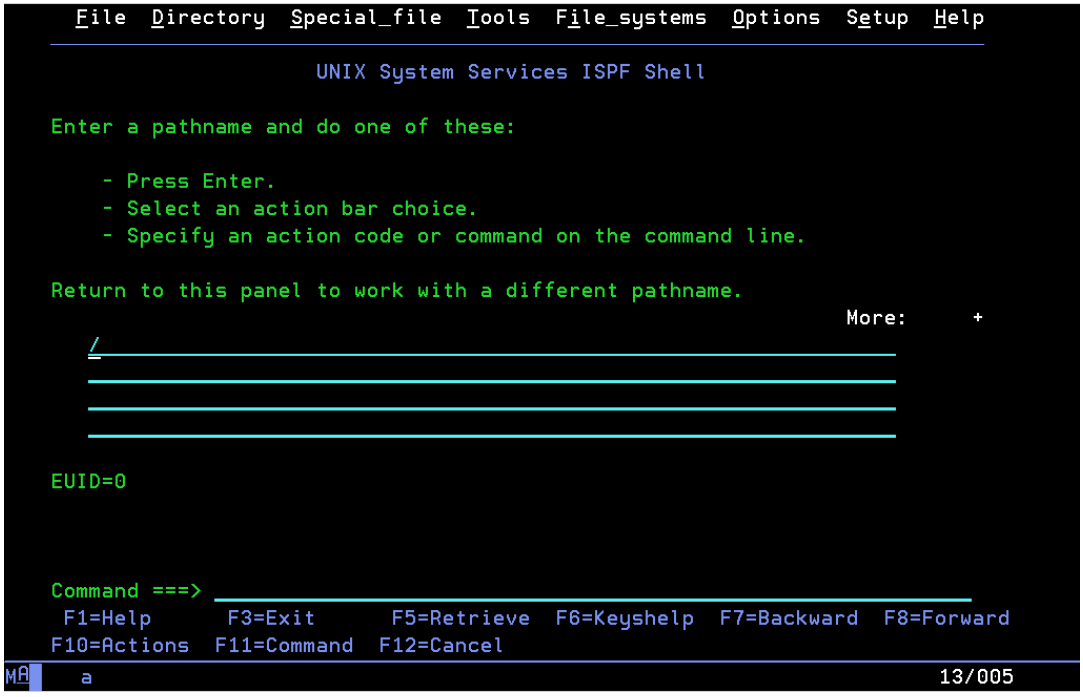


Figura 2.33 Pantalla inicial de ISH.

Una vez en esta pantalla situamos el cursor sobre el menú Tools de la parte superior y pulsamos Control para que este se despliegue. La pantalla con el menú desplegado quedará como podemos ver en la Figura 2.34.

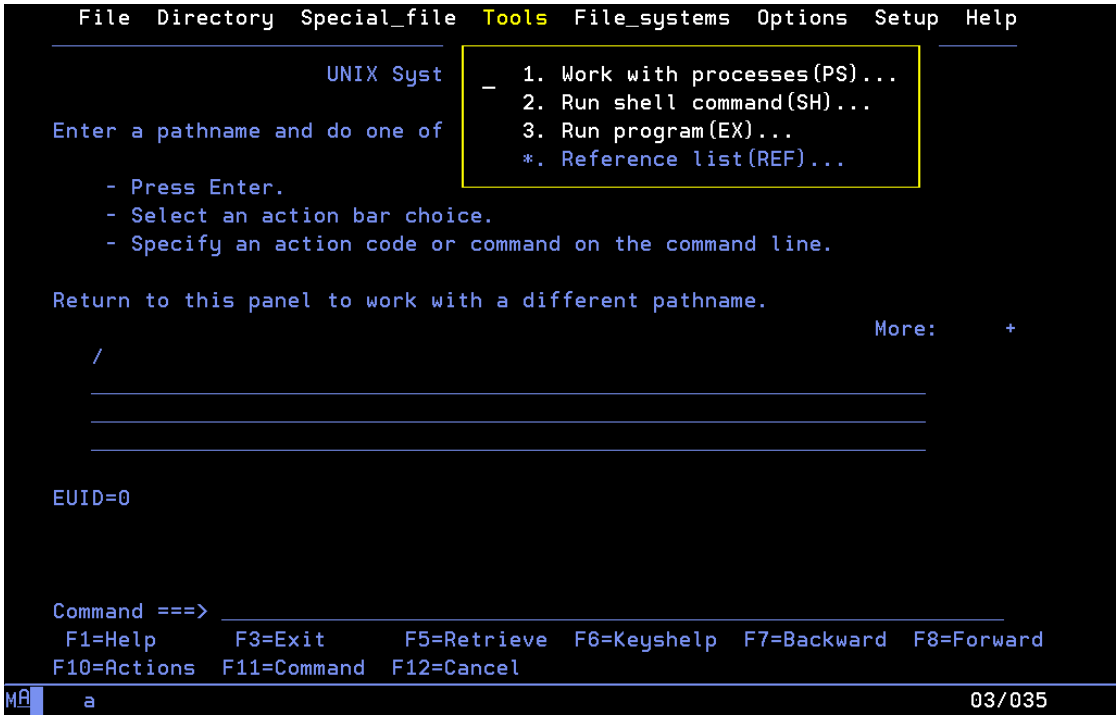


Figura 2.34 Pantalla de ISH con menú Tools desplegado

Una vez que hayamos desplegado el menú Tools escribimos 2 y pulsamos Control para abrir la consola de comandos de ISH. La consola de comando tiene la apariencia que podemos ver en la Figura 2.35.

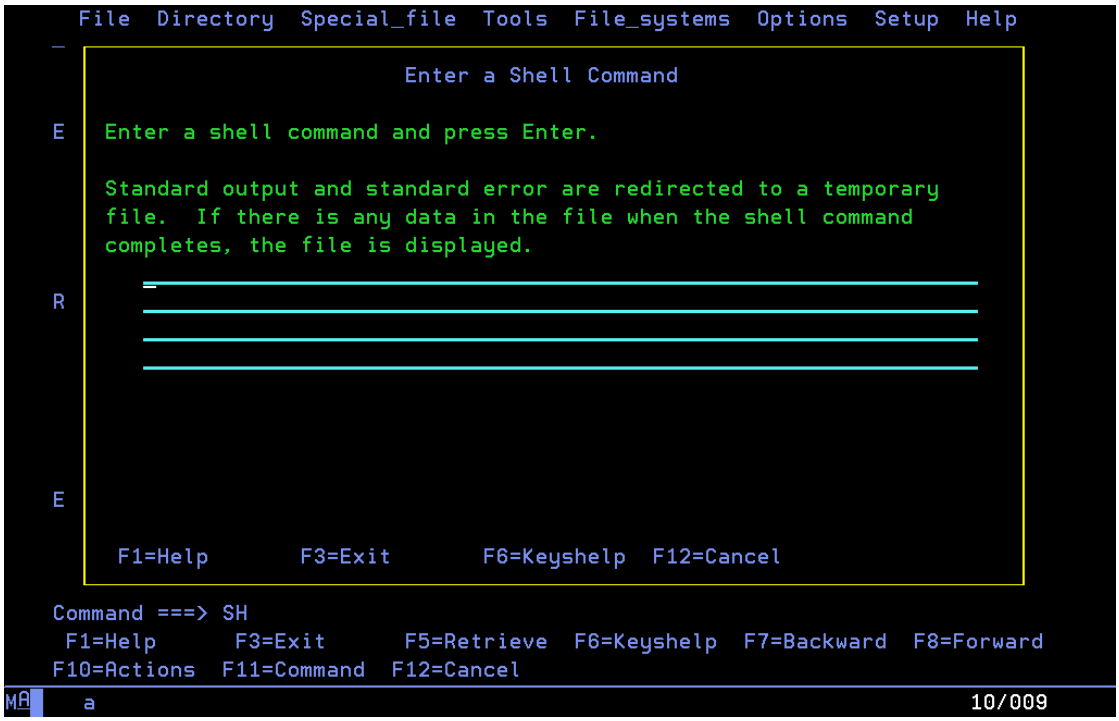


Figura 2.35 Consola de comandos de ISH

Una vez llegados a este punto ya estamos listos para crear los certificados tanto de la máquina cliente como del mainframe.

Los pasos necesarios para hacer esto son los siguientes:

1.- En el mainframe, creamos el key ring y el certificado auto-firmado del servidor por medio del comando:

```
keytool -genkey -alias pruebasCert -keysize 1024 -dname
"cn=pruebas.es,o=ORG,ou=ORG1,l=PAMPLONA,s=NAVARRA,c=SPAIN" -keystore
/etc/certificados/pruebasCert.jks –keypass pruebasPass -storepass pruebasPass
-keyalg RSA
```

Donde los parámetros del comando significan:

- alias → nombre que identifica el certificado auto-firmado y la clave privada
- keysize → tamaño de la clave
- dname → nombre X.500 asociado al alias
  - cn → nombre común
  - o → organización
  - ou → unidad de la organización
  - l → ciudad
  - s → provincia
  - c → país
- keystore → localización del key ring, lo crea si no existe. El fichero tiene extensión .jks
- keypass → contraseña usada para proteger la clave privada. Tiene que ser la misma que la storepass para permitir que el CTG establezca una conexión sobre SSL
- storepass → contraseña del keyring
- keyalg → algoritmo utilizado para generar el par de claves

2.- Podemos listar los certificados contenidos en el key ring por medio del comando:

```
keytool -list -keystore /etc/certificados/pruebasCert.jks -storepass
pruebasPass -v
```

3.- Exportamos el certificado auto-firmado del servidor a un fichero arm con el comando:

```
keytool -export -alias pruebasCert -keystore
/etc/certificados/pruebasCert.jks -storepass pruebasPass -file
/etc/certificados/pruebasCert.arm -rfc
```

Donde los parámetros significan:

- file → el nombre del fichero al que vamos a exportar el certificado. Tiene extensión .arm
- rfc → exporta el certificado en formato RFC

Obtendremos el siguiente mensaje de confirmación:
`Certificate stored in file </etc/certificados/pruebasCert.arm>`

4.- Transferimos el fichero .arm con el certificado a nuestra máquina usando el BlueZone Secure FTP en modo **binario**.

5.- En nuestro PC, abrimos una consola de comandos e importamos el certificado del servidor por medio del comando:

```
keytool -import -alias pruebasCert -file C:\Certificados\pruebasCert.arm -
keystore C:\Certificados\nuestraMaquinaCert.jks -storepass nuestraMaquinaPass
–noprompt
```

Donde los parámetros significan:

- alias → alias que le hemos dado al certificado del servidor al crearlo en el paso 1
- file → ruta del fichero en nuestra máquina donde hemos transferido el fichero .arm
- keystore → ruta donde creamos el key ring en la máquina cliente
- storepass → contraseña que le damos al key ring del cliente. No tiene que ser la misma que la del servidor
- noprompt → elimina la necesidad de confirmar que el certificado va a ser importado

6.- Creamos el certificado del cliente por medio del comando:

```
keytool -genkey -alias nuestraMaquinaCert -keysize 1024 -dname
"cn=nuestraMaquina.es,o=ORG,ou=ORG1,l=pamplona,s=navarra,c=spain" -keystore
C:\Certificados\nuestraMaquinaCert.jks -keypass nuestraMaquinaPass -storepass
nuestraMaquinaPass -keyalg RSA
```

7.- Exportamos el certificado del cliente a un fichero .arm con el comando:

```
keytool -export -alias nuestraMaquinaCert -keystore
C:\Certificados\nuestraMaquinaCert.jks -storepass nuestraMaquinaPass -file
C:\Certificados\nuestraMaquinaCert.arm –rfc
```

8.- Transferimos el .arm por FTP en modo **binario** al host usando BlueZone Secure FTP.

9.- Importamos el certificado del cliente al key ring del host con el comando:

```
keytool -import -alias nuestraMaquinaCert -file
/etc/certificados/nuestraMaquinaCert.arm -keystore
/etc/certificados/pruebasCert.jks –storepass pruebasPass –noprompt
```

Con esto ya hemos creado los certificados del cliente y el servidor necesarios para establecer una conexión sobre SSL. Para que el CTG admita este tipo de conexiones, en el fichero de configuración .ini del mismo añadimos en la SECTION GATEWAY las siguientes líneas, o descomentamos las que hay por defecto y las modificamos.

```
protocol@ssl.handler=com.ibm.ctg.server.SslHandler
protocol@ssl.parameters=port=8050;\
                        connecttimeout=2000;\
                        idletimeout=600000;\
                        pingfrequency=60000;\
                        keyring=/etc/certificados/pruebasCert.jks;\
                        keyringpw=pruebasPass;\
                        clientauth=off;
```

El parámetro *keyring* indica la ruta donde está almacenado el key ring en el servidor con los certificados del cliente y el servidor.

El parámetro *keyringpw* indica la contraseña que protege al key ring.

Paramos y volvemos a arrancar el CTG para que recoja los cambios realizados.

2.7.2.- Realizar peticiones sobre SSL desde Java

Una vez que hemos realizado los pasos anteriores ya podemos hacer peticiones ECI sobre SSL. Las peticiones SSL sólo las podemos hacer desde aplicaciones Java.

Los cambios que hay que hacerles a las aplicaciones para realizar peticiones sobre Java no son demasiadas.

Añadimos la siguiente línea para poder hacer uso de la clase necesaria para indicar donde está el key ring con los certificados y su contraseña.

```
import java.util.Properties;
```

Creamos una variable de la clase que acabamos de importar y añadimos las propiedades que necesitamos para poder usar SSL.

```
Properties propiedadesSSL;
propiedadesSSL = new Properties();
propiedadesSSL.setProperty(JavaGateway.SSL_PROP_KEYRING_CLASS,
"C:\\Certificados\\nuestraMaquinaCert.jks");
propiedadesSSL.setProperty(JavaGateway.SSL_PROP_KEYRING_PW,
"nuestraMaquinaPass");
```

Pasamos esta variable a la hora de crear la JavaGateway de forma que se establezca una conexión sobre SSL.

```
jg = new JavaGateway(URL, puerto, propiedadesSSL);
```

Con esto ya podemos hacer peticiones sobre SSL. Para ello, la forma de la URL donde está el CTG que le pasemos para crear la JavaGateway será de la forma ssl://<IP_del_CTG>. El puerto, si hemos dejado el que viene por defecto, será el 8050.

Si no introducimos correctamente alguna de las propiedades necesarias para establecer la conexión sobre SSL obtendremos el siguiente mensaje de error:

```
CTG6672E Una o más propiedades del protocolo SSL no se han definido
```

Si los valores de la ruta donde está el key ring o su contraseña son inválidos también nos lo indica.

Si realizamos una petición por SSL y comprobamos en el SDSF la información que nos suministra el CTG, vemos lo siguiente que indica que la petición ha venido por SSL:

```
CTG6506I Client connected: ÝConnectionManager-0~ -
ssl@7bf47bf4ÝSSL_RSA_WITH_RC4_128_MD5:
SocketÝaddr=/10.238.12.77,port=1789,localport=8050~~ using protocol TLSv1
```

Además, capturamos por medio de un *sniffer* como Wireshark los paquetes que se intercambian entre nuestra máquina y el CTG, por ejemplo al hacer una petición al programa EC01 de ejemplo que devuelve la fecha y hora del mainframe.

Si la petición la hacemos sobre TCP, podemos ver el siguiente stream intercambiado entre nuestra máquina y el CTG donde podemos ver los datos que se intercambian:

```
Gate.p.................BASE.......es..ES.............Gate.p.................
BASE....;..en..US...%OS: z/OS Arch: s390 Version:1.08.00.........Gate.p.....
............ECI.........................CICS1.............................EC0
1...............P.........Gate.p.................ECI....j.....................
...P.02/02/11 12:59:15..Gate.p ................BASE.....Gate.p..............
..BASE.....
```

Podemos ver cosas como la versión del z/OS, el CICS al que hacemos la petición o los datos que nos devuelve

En cambio, si hacemos la misma petición pero sobre SSL vemos el siguiente stream:

```
.b....9............../..3..2................@.....................MID..=N.[h,
iy.&......D(....KI14).........F..MIHqh..HP....}<.Y.Z...(Q....N..MIHq....7%$...
j......FI.......1......{..x..u0..q0..........MG..0..*.H.......0}1.0...U....SP
AIN1.0...U....NAVARRA1.0...U....PAMPLONA1.0..U....PIN1.0...U...ORG1$0"..U....
pruebas.es0..110201111728Z.110502111728Z0}1.0...U....SPAIN1.0...U....NAVARRA1
.0...U....PAMPLONA1.0..U....ORG1.0...U...ORG1$0"..U....pruebas.es0..0..*.H...
.........0.......1...N....v....#..^.M.e..3V...GY..=...]...w"QS.....gY.........
_....g~.p3#sX..;.Iu...?.q.....X...*%...G..c..nL....%OBS.k....._.....0..*.H...
.........Z1.5}.../....Pc'..../.:..s..~.<T..]....N@...c....>...c..@..=...$.A..
..o3.W)r_9...n.*(.z..q........-.....q....M..`...>....)1..............l..xig
Y(.....q=w.a..{._D...w4...X.ef.....Aa.^...=.1?~..........HSr..+........./....
..........N:N....l..G`fk..v1Z&..@*..3.n..a.............!.3...T...%h.V.G..6E.
o....O."...........!J.S9R^....O..A....Z.4`...(.K......G.Z.t.T.]....%.f2.MM.x5
.wu....L8.......O.4.I..Ge9.Ft;(.<.Y..;....L..#......lEDG.W...Q*.;.{f...!.....
./..$...........?j.........Z|?[.5Qw.~Y..[.EED..8..)}.f....G.....>..G......g.
.l.........51...&zH.v]u....&mP./.G..VQ..[.._.r..-.y...f.X..?/..y$H..l_....s.~
Mt.!.o.p....d.t.....!.A9B...*...7...b..J.%]....~.'...7y!.......`.e..b.2%.....
3.((V.,o.4.(z....rtL..uIR.A.t[........3u$...ms...z...C.......^}TH.W.MH...Z...
..2............3?,...1TV.C.EL.....l..%....gU+T........_BdX....).=..L....8.TV%
z.M..37../..../u.'7...X'...!....!Q..(.=.pl.!.A-.dB.^'..._......4id..4...I.1...
```

```
....1."..._QP.Q....:|.=...0....Xp.].....<..(f{'.t_.X.3.....1.....M`t..o+V.Se.2
.I....Hx..a..._......E._..x..........1..t.........b.......A...0Ov....N=Y..c
```

Ahora no podemos ver ningún dato legible excepto parte de los datos que hemos introducido al crear el certificado que sirve para autentificar al servidor. De esta forma comprobamos como los datos parecen estar encriptados.

2.8.- Port Sharing

A fin de asegurar un mayor rendimiento y disponibilidad del CTG podemos configurar dos, o más, CTGs en lo que se conoce como *Port Shared*. Se configuran dos, o más, CTGs que escuchan peticiones en un mismo puerto, en este caso el 2006. Cuando llegan peticiones, cada CTG atiende alternativamente una petición. En caso de caer un CTG por cualquier razón, los demás atienden las peticiones sin que el usuario note nada. En caso de caer todos los CTGs ecepto uno, este se encargará de atender todas las peticiones. No será hasta que caigan todos los CTGs cuando el usuario recibirá un error que le indique que no es posible conectarse a un CTG en la IP y puerto indicado. Todos los CTGs que se encuentren en un mismo grupo de *Port Shared* deben estar instalados en la misma LPAR. Los CTGs configurados de esta manera se dice que están en *Port Sharing* y su uso es transparente para el usuario.

En la Figura 2.36 podemos ver dos CTGs configurados en *Port Sharing*.

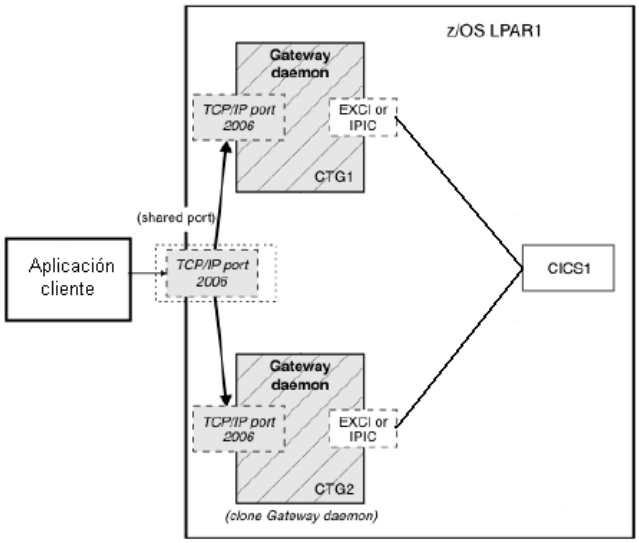


Figura 2.36 Dos CTGs configurados en *Port Sharing*

Para poder configurar dos, o más, CTGs en *Port Sharing*, el puerto donde van a escuchar las peticiones, en este caso el 2006, tiene que estar reservado como SHAREPORT en el miembro PROFILE de la librería PRUEBAS.TCPIP.PARMLIB. Esto ya lo habíamos hecho en previsión de realizar esta configuración de CTGs.

Para configurar dos, o más, CTGs en *Port Sharing* tenemos que tener una copia de los siguientes elementos por cada uno de los CTGs que vamos a configurar en *Port Sharing*:

- Archivo de inicialización .ini
- Miembro con las variables de entorno
- Procedimiento de arranque
- Definición del recurso TCPIPSERVICE

- Definición del recurso IPCONN si queremos habilitar la seguridad

Vamos a configurar dos CTGs en *Port Sharing* por lo que hacemos una copia de cada uno de estos elementos. Los nombres de los elementos copiados serán los mismos que los de los elementos originales excepto que cambiamos la A por la B para indicar que se refieren al CTG B del grupo 1 de CTGs en *Port Sharing*.

La configuración de todos los elementos será idéntica exceptuando los siguientes cambios que hay que realizar:

- Archivo de inicialización .ini
  - Applid del CTG
    - Lo cambiamos por su valor correspondiente que en este caso será CTG1B
  - Parámetro puerto de la SECTION IPICSERVER
    - Le damos el valor 20066. Los puerto del 20061 al 20066 los reservamos para conexiones del CTG1A a más CICS
- Miembro con las variables de entorno
  - Variable CICSCLI
    - Le damos el valor correspondiente para que apunte al archivo .ini adecuado. En este caso será /etc/ctgvar/ctg1b.ini
  - Variable CTG_RRMNAME
    - Le damos el valor adecuado que será CTG1B.IBM.UA
- Procedimiento de arranque
  - Variable CTGUSR
    - Le damos el valor correspondiente para que apunte al miembro de variables de entorno adecuado que será CICSTG.V7R2M0.CTGENVAR(ENVCTG1B)
- Definición del recurso TCPIPSERVICE
  - Parámetro Netname
    - Le damos el Applid del CTG correspondiente que será CTG1B
  - Parámetro POrtnumber
    - Indicamos el puerto por donde se va a conectar al CICS que será el 20066
- Definición del recurso IPCONN
  - Parámetro Applid
    - Le damos el Applid del CTG correspondiente que será CTG1B
  - Parámetro Tcpipservice
    - Le indicamos el TCPIPSERVICE asociado que será CTG1B

Arrancamos el CTG1B que acabamos de configurar y hacemos varias peticiones al puerto 2006. Podemos ver en la información que suministra el CTG en el SDSF como cada petición se dirige alternativamente a uno y otro CTG.

Si paramos uno de los CTGs y realizamos peticiones, el otro las atiende correctamente de forma transparente para el usuario.

2.9.- Rendimiento

Una vez que tenemos el CTG instalado y en funcionamiento, hay que tener en cuenta diferentes factores que pueden afectar al rendimiento del CTG y algunas medidas que podemos tomar para mejorarlo.

No sólo el CTG afecta al rendimiento total del sistema sino que necesitamos entender como afectan todos los componentes del sistema al rendimiento.

Estos componentes pueden incluir:

- Navegador web
- Router y firewalls
- Servidor de aplicaciones web, servidor de aplicaciones J2EE o máquina donde se ejecuta la aplicación de escritorio
- CICS Transaction Gateway
- CICS

Vamos a recolectar información sobre:

- Carga del procesador
- Ratios de transferencia de datos
- Tiempos de respuesta

Los tiempos de respuesta son indicadores particularmente útiles ya que nos ayudan a entender que componentes del sistema afectan más al rendimiento.

Para evaluar el rendimiento del CTG y del sistema tenemos que entender que el CTG proporciona un modelo multi-hilo para manejar las conexiones de red y para asignar hilos a las peticiones y respuestas de los clientes remotos. Este modelo usa fundamentalmente los siguientes objetos:

- Hilos administrador de conexiones
  - Un hilo administrador de conexiones maneja todas las conexiones de un cliente remoto particular. Cuando recibe una petición, asigna un hilo de trabajo del grupo de hilos de trabajo disponibles para ejecutar la petición
- Hilos de trabajo o hilos trabajadores
  - Un trabajador es el objeto que ejecuta una petición de un cliente remoto. Cada trabajador tiene su propio hilo que se activa cuando hay algún trabajo que hacer. Cuando un hilo de trabajo finaliza la ejecución vuelve al conjunto de hilos de trabajo disponibles.

En la Figura 2.37 podemos ver este modelo.

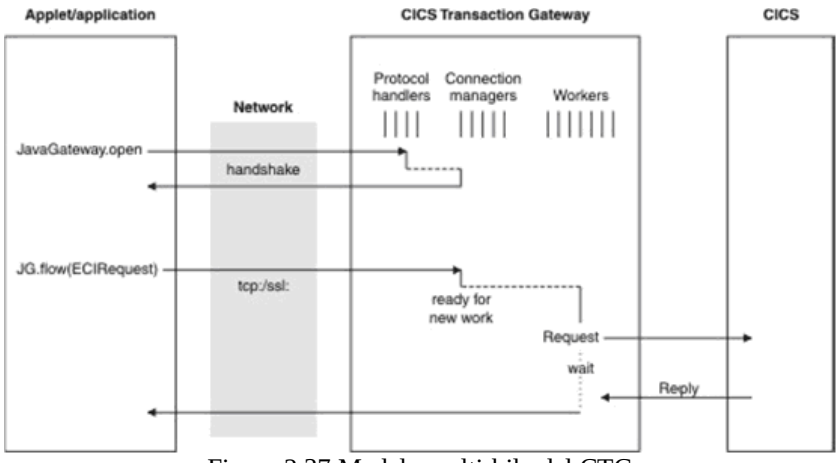


Figura 2.37 Modelo multi-hilo del CTG

2.9.1.- Obtener estadísticas

Para poder comprobar el rendimiento del CTG en muchos casos necesitaremos consultar las estadísticas que nos ofrece el CTG. Esto lo podemos hacer desde una sesión de TSO por medio del SDSF o desde una aplicación Java realizando una petición ECI.

2.9.2.1.- Estadísticas ofrecidas por el CTG

El CTG ofrece 9 grupos de estadísticas que son:

- LS : Estadísticas sobre los servidores CICS lógicos. SECTION LOGICALSERVER del archivo .ini
- LSx : Estadísticas sobre el servidor CICS lógicos x
- CM : Estadísticas sobre los hilos administradores de conexiones
- CS : Estadísticas sobre todos los servidores CICS
- WT : Estadísticas sobre los hilos trabajadores
- CSx : Estadísticas sobre el servidor CICS x. Un grupo por cada servidor CICS
- SE : Estadísticas sobre el entorno del sistema
- PH : Estadísticas sobre los protocolos
- GD : Estadísticas sobre recuento de transacciones, peticiones y estado del CTG

Algunas estadísticas se muestran dos veces, una precedida de la letra L y otra de la letra I. La primera se refiere al total desde que se arranco el CTG mientras que la segunda se refiere al intervalo actual. En nuestro caso, un intervalo dura 3 horas.

Hay muchas estadísticas y la mayoría de ellas no nos van a influir en el rendimiento. Aquellas que vamos a utilizar para comprobar el rendimiento de nuestro sistema son las siguientes:

- CM_IALLOCHI Número máximo de hilos administradores de conexiones asignados al mismo tiempo a aplicaciones cliente
- CM_SMAX Número máximo de administradores de conexiones que puede haber en el sistema
- CS_IAVRESP Tiempo medio de respuesta del CICS, en milisegundos
- WT_CCURR Número actual de hilos trabajadores creados
- WT_IALLOCHI Número máximo de hilos trabajadores asignados al mismo tiempo a hilos administradores de conexiones
- WT_ITIMEOUTS Número de veces que se ha alcanzado el timeout de los hilos trabajadores
- WT_SMAX Número máximo de hilos trabajadores
- CSx_IAVRESP Tiempo medio de respuesta del CICS x, en milisegundos
- SE_CELOAL Cantidad de memoria usada por el CTG
- SE_CHEAPGMIN Tamaño del Java heap después del último evento del Garbage Collector. Más adelante se explica que es el Java heap
- SE_IGCCOUNT Número de eventos del Garbage Collector en este intervalo
- SE_IGCTIME Tiempo en milisegundos que ha estado funcionando el Garbage Collector en este intervalo
- SE_SELIM Cantidad máxima de memoria del CTG
- SE_SHEAPMAX Tamaño máximo de Java heap

- GD_IAVRESP Tiempo medio de respuesta en milisegundos del CTG en este intervalo
- GD_IREQDATA Cantidad de datos enviados por los clientes en este intervalo
- GD_IRESPDATA Cantidad de datos enviados a los clientes en este intervalo
- GD_IRUNTIME Tiempo de ejecución de este intervalo
- GD_LAVRESP Tiempo medio de respuesta en milisegundos del CTG en total

La totalidad de las estadísticas se puede consultar en el manual SC34-6961 - CTG z-OS Administration que se puede descargar desde la página oficial de IBM.

2.9.2.2.- Estadísticas desde SDSF

Para obtener las estadísticas del CTG tenemos la opción de hacerlo por medio del SDSF en una sesión TSO o desde una aplicación Java, no se puede hacer desde una aplicación .Net.

Para obtener las estadísticas desde SDSF, nos situamos en el panel de Log para lo que, desde el panel principal de TSO, introducimos SD y a continuación LOG. Esto nos lleva al panel que podemos observar en la Figura 2.38.

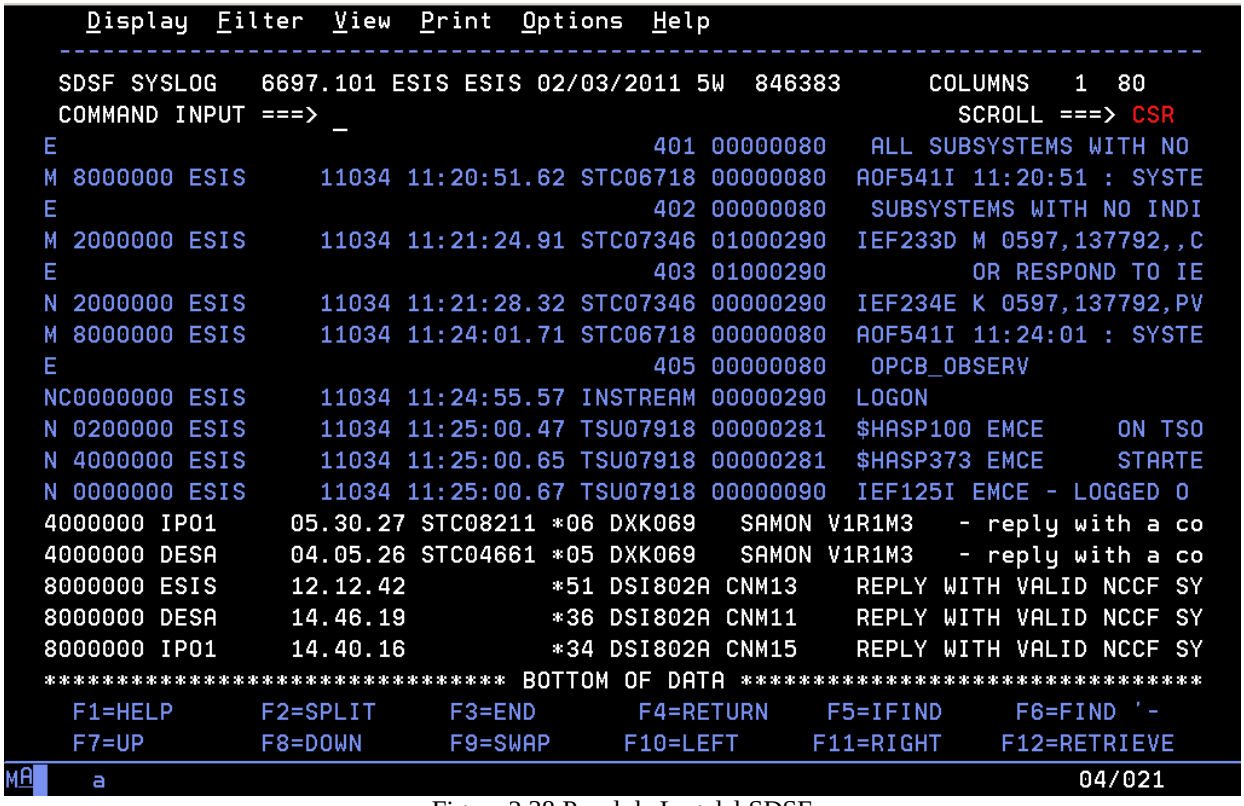


Figura 2.38 Panel de Log del SDSF

En la parte superior, vemos que hay una línea que pone *COMMAND INPUT*. En esta línea vamos a escribir los comandos que nos va a permitir obtener las estadísticas. Para obtener estadísticas, introducimos uno de los siguientes comandos:

- /F *nombre_ctg*,APPL=STATS,GS
- /F *nombre_ctg*,APPL=STATS,GS=*grupo_estadísticas*

El primero nos muestra todas las estadísticas mientras que al segundo le podemos decir que grupos de estadísticas queremos que nos muestre separados por ":". Por ejemplo, si quisiéramos los grupos WT y CM introduciríamos
/F *nombre_ctg*,APPL=STATS,GS=WT:CM

No podemos pedir estadísticas individuales, solamente grupos de estadísticas.

En la Figura 2.39 podemos ver un ejemplo de los que se muestra en el Log al introducir el comando para listar todas las estadísticas.

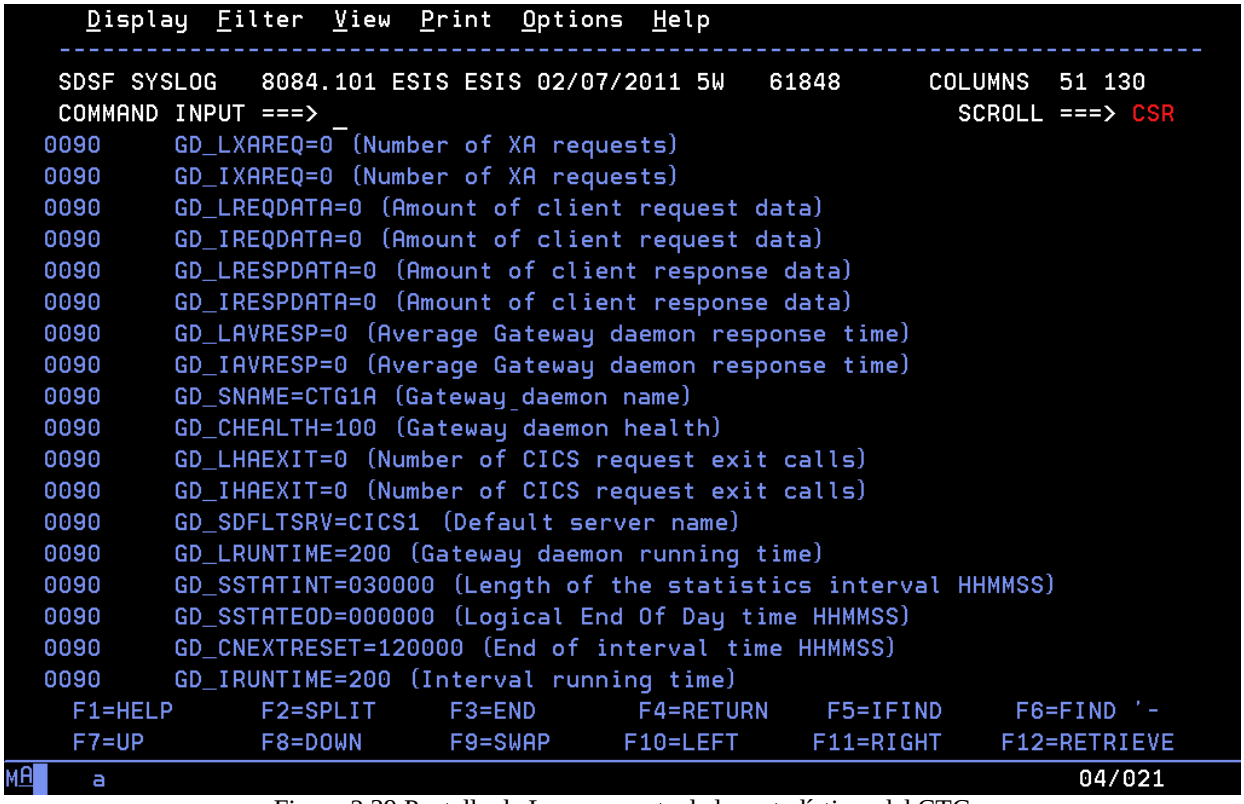


Figura 2.39 Pantalla de Log con parte de las estadísticas del CTG

2.9.2.3.- Estadísticas desde Java

La otra forma que tenemos de obtener estadísticas es desde una aplicación escrita en Java. Para ello, lo primero que tenemos que hacer es habilitar un puerto en el CTG para que pueda recibir peticiones de estadísticas. Para hacer esto, en el archivo de configuración .ini añadimos las siguientes líneas, o descomentamos las que vienen por defecto:

```
protocol@statsapi.handler=com.ibm.ctg.server.RestrictedTCPHandler
protocol@statsapi.parameters=connecttimeout=2000;\
port=2980;\
bind=;\
maxconn=5;
```

El parámetro más importante es el *port* ya que indica en que puerto va a escuchar el CTG las peticiones de estadísticas. Para habilitar los cambios tendremos que parar y volver a arrancar el CTG si ya estaba en funcionamiento.

Una vez hecho esto, ya podemos realizar peticiones de estadísticas desde una aplicación Java. Para ello, en vez de usar la librería ctgclient.jar que estábamos usando hasta ahora para hacer las

peticiones ECI, vamos a usar la librería ctgstats.jar que también se suministra con el CTG. El paquete que tenemos que importat es el com.ibm.ctg.client.stats.*. En el Documento 2.9 podemos ver un ejemplo de código Java que hace una petición de todas las estadísticas y las muestra.

```
GatewayConnection gc = null;
String URL = "20.20.20.1";
Inr puerto = 2980;
try{

	gc = new GatewayConnection(URL, puerto);

	StatResultSet srs = null;
	try {
		srs = gc.getStats();
	} catch (Exception e) {
		System.out.println("Error: Fallo al recolectar estadisticas.\n");
		System.exit(0);
	}

	String st = "";

	String vr = gc.getStatsAPIVersion();
	st = "API Version: "+vr+"\n####################\n\n";

	for (StatData stat : srs){
		if (stat.getRc() == StatData.SUCCESSFUL_QUERY) {
			st = st
			+ stat.getStatId()
			+ "="
			+ stat.getValue()
			+ " ("
			+ stat.getShortDesc()
			+")\n";
		} else {
			System.out.println("Peticion para "
			+stat.getQueryElement()
			+" fallo con "
			+stat.getReturnString(stat.getRc())
			+ "\n");
		}
	}

	System.out.println(st);

}catch(Exception ex){
	System.out.println("Error\n" + ex.getMessage());

}finally{
	gc.close();
}
```

Documento 2.9 Código Java para pedir todas estadísticas al CTG

Si ejecutamos esta aplicación obtenemos las mismas estadísticas y valores que los obtenidos al hacer la petición desde el SDFS. En este caso si podemos pedir estadísticas por grupos o estadísticas individuales, separadas por ":" en caso de querer pedir más de una estadística o grupo.

2.9.2.- Medidas de rendimiento

Hay varios factores que influyen en el rendimiento del CTG. Algunos de ellos se pueden medir por medio de las estadísticas. Otros no se pueden medir y simplemente los tenemos que tener en cuenta a la hora de diseñar y configurar nuestro sistema.

2.9.2.1.- Hilos administradores de conexiones

- Si el valor especificado para el **Número inicial de hilos administradores de conexiones**, parámetro *initconnect* del archivo .ini, es muy alto nuestro sistema va a malgastar recursos administrando hilos que no son necesarios
- Si el valor de **Número máximo de hilos administradores de conexiones**, parámetro *maxconnect* del archivo .ini, es muy bajo para satisfacer todas las solicitudes, cada nueva petición puede necesitar esperar que un hilo quede disponible. Si el tiempo de espera supera el valor especificado en el parámetro *connecttimeout* en la definición del protocolo en el archivo .ini el CTG deniega la conexión

El diseño de las aplicaciones determina el número de hilos administradores de conexiones que vamos a necesitar. Las conexiones entrantes al CTG pueden venir, por ejemplo, de un servlet donde cada copia del servlet emite sus propias peticiones ECI pero comparten un solo hilo administrador de conexiones. Por otra parte la aplicación puede tener un conjunto de conexiones y cada petición ECI ser lanzada por una conexión del conjunto.

El CTG crea un nuevo hilo administrador de conexiones y una nueva conexión TCP/IP cada vez que una aplicación cliente Java crea un objeto JavaGateway. Esto significa que el rendimiento del sistema puede ser mejor si lanzamos varias peticiones ECI usando el mismo objeto JavaGateway, y por el mismo hilo, en vez de crear un nuevo objeto JavaGateway para cada petición.

Emitir múltiples peticiones a través del mismo objeto JavaGateway también reduce los recursos de sistema necesarios para crear y destruir los objetos JavaGateway.

2.9.2.2.- Hilos trabajadores

Los hilos trabajadores manejan las conexiones entre el CTG y el servidor CICS. Dependiendo del diseño de nuestras aplicaciones y de la carga de trabajo necesitaremos más o menos hilos trabajadores. Cuanto mas tiempo permanezcan en procesamiento nuestras transacciones, más hilos trabajadores necesitaremos para mantener un ratio de transacciones finalizadas determinado.

- Si el valor especificado para el **Número inicial de hilos trabajadores**, parámetro *initworker* del archivo .ini, es muy alto el CTG usará recursos para manejar hilos que no necesita
- Si el valor es muy bajo, el CTG usará recursos para buscar hilos trabajadores que estén disponibles
- No se puede abrir más conexiones EXCI que el máximo impuesto por CICS. Esto significa que no podemos mejorar el rendimiento especificando un valor mayor que este impuesto por CICS para el **Número máximo de hilos trabajadores**, parámetro *maxworker* del archivo .ini.

2.9.2.3.- Mostrar nombres TCP/IP

Seleccionar esta opción, parámetro *noname=off* en el archivo .ini, puede causar un descenso en el rendimiento de algunos sistemas.

2.9.2.4.- Valores de timeout

Es poco probable que se mejore el rendimiento cambiando los valores de timeout aunque en algunos casos puede ser necesario para ciertas aplicaciones.

Si la ejecución de un programa utiliza mucho tiempo en finalizarse, es posible que se necesite aumentar este valor para que pueda finalizar en tiempo y no se produzca un error.

Si este valor es muy alto, y por lo que sea se produce algún suceso que hace que el programa no finalice, por ejemplo por un bucle, y hemos enviado la petición en forma síncrona, tendremos que esperar más a que se produzca el timeout para tener de nuevo el control de nuestra aplicación.

Esto quiere decir que no es mejor ni peor tener un valor de timeout alto o bajo. Depende de lo que queremos hacer.

2.9.2.5.- *Logging de conexiones*

La configuración del Gateway puede controlar si el CTG va a escribir un mensaje cada vez que una aplicación cliente se conecte o desconecte. Esta opción puede reducir significativamente el rendimiento especialmente si se producen conexiones o desconexiones frecuentemente.

2.9.2.6.- Tamaño máximo del área de almacenamiento de Java o Java heap

El Java heap es el área de almacenamiento donde Java almacena los objetos y tipos primitivos compartidos creados.

Si nuestro sistema necesita un gran número de hilos administradores de conexiones podemos necesitar aumentar el tamaño del área de almacenamiento para mejorar el rendimiento. Como hacer esto depende de la JVM y se debe consultar la documentación adecuada para cada JVM para obtener más información.

En general se puede hacer por medio del comando

java –Xms<*tamaño_inicial*>m –Xmx<*tamaño_maximo*>m

La primera opción establece el tamaño inicial, en megas, del Java heap. La segunda opción establece el tamaño máximo del Java heap.

2.9.2.7.- Objetos JavaGateway

El rendimiento es mejor si lanzamos varias peticiones usando el mismo objeto JavaGateway que si creamos un nuevo objeto JavaGateway cada vez que queremos mandar una petición. Cada vez que creamos y destruimos un objeto JavaGateway necesitamos recursos de sistema adicionales para:

- Crear y destruir el objeto en si
- Crear y destruir sus sockets asociados
- Gasto en el uso del recolector de basura de Java o Garbage Collector

2.9.2.8.- Tamaño de la COMMAREA

El tamaño de la COMMAREA tiene un efecto importante en el rendimiento. Si utilizamos grandes COMMAREAs necesitamos más recursos para procesarlas y los tiempos de respuesta se vuelven más grandes. Además, toda la COMMAREA se envía y recibe cada vez, independientemente de que tenga datos o no, lo que hace que los tiempos de transmisión de los datos aumente.

Hay que intentar usar un tamaño de COMMAREA lo más ajustado posible a los datos que vamos a utilizar.

2.9.2.9.- Llamadas ECI síncronas o asíncronas

CTG tiene que hacer menos procesamiento para manejar llamadas ECi síncronas en vez de su equivalente asíncrona. También las llamadas síncronas crean menos flujo de red que las llamadas asíncronas. Esto significa que este tipo de llamadas mejoran el rendimiento del sistema.

2.9.2.10.- Trazas

No se recomienda el uso de la traza completa del CTG para monitorizar el rendimiento en un entorno de producción. Esto es debido a que las trazas disminuyen significativamente el rendimiento.

Cuando es posible hay que tratar de medir los tiempos de respuesta a través de diferentes partes del sistema sin utilizar trazas para encontrar donde se producen los mayores retrasos. Por ejemplo, podemos medir los tiempos de respuesta en la aplicación cliente y también a través del servidor de aplicaciones o el CICS.

2.9.2.11.- Tamaño de la región y condiciones de "out of memory"

Si nuestras aplicaciones necesitan que el CTG cree un gran número de hilos administradores de conexiones o trabajadores es posible que sea necesario incrementar el tamaño de la región, o cantidad de memoria que puede usar el CTG. El tamaño de la región se define con el parámetro REGION en el procedimiento de arranque del CTG.

Hay que tener en cuenta que un valor REGION=0M, similar a memoria infinita, puede producir una excepción java.lang.OutOfMemory en algunos sistemas por lo que no es aconsejable usarlo.

Podemos usar las estadísticas del CTG para establecer si la cantidad de almacenamiento virtual disponible que está siendo usado por el CTG excede la cantidad de memoria disponible para el mismo.

<u>Estadísticas a considerar</u>

- CM_SMAX
- SE_SELIM
- SE_CELOAL
- SE_SHEAPMAX
- WT_SMAX

El almacenamiento virtual usado por el espacio de direcciones del CTG, mostrado en la estadística SE_CELOAL, no puede exceder el máximo de memoria disponible, estadística

SE_SELIM. Hay tres factores que afectan los requerimientos de almacenamiento virtual del espacio de direcciones del CTG:

- El área máxima de Java heap, SE_SHEAPMAX. El valor por defecto es 128MB; normalmente esto es suficiente para la mayoría de las cargas de trabajo si usamos los parámetros de configuración por defecto para los hilos
- La pila de almacenamiento usada por los hilos del CTG; esto es un total de 512KB por hilo
- El almacenamiento necesario para que el CTG funcione; esto es aproximadamente 50MB por CTG pero varía si usamos funcionalidades adicionales como XA, grabación de estadísticas por SMF, SSL, etc.

Si un CTG intenta crear un nuevo hilo y la pila de almacenamiento necesita más espacio virtual del que está disponible se produce una condición de *out of memory*; es probable que esto cause la finalización del CTG. Podemos usar el siguiente cálculo para evitar estas condiciones. Estima el requerimiento máximo potencial de almacenamiento virtual del CTG. El cálculo da un resultado en KB:
51200 + (SE_SHEAPMAX / 1024) + (512 + (WT_SMAX + CM_SMAX))

Según la configuración que hemos hecho tendríamos el siguiente cálculo:
51200 + (134217728 / 1024) + (512 (100 + 100)) = 284672KB = 278MB

Este valor, algo más por utilizar funcionalidades como SSL, sería el que tendríamos que darle al parámetro REGION en el procedimiento de arranque del CTG.

Un gran número de hilos totales, más de 200, o el uso de conexiones IPIC puede resultar en una mayor demanda de asignación de Java heap. Incrementar el tamaño máximo del heap a 256MB debe ser adecuado para la mayoría de las instalaciones.

2.9.2.12.- Tiempos de respuesta mediditos con estadísticas

Diferentes configuraciones del sistema y cargas puede llevar a malos tiempos de respuesta. Podemos usar las estadísticas que provee el CTG para identificar posibles causas de los malos tiempos de respuesta y mejorarlos.

A continuación podemos ver algunos consejos para mejorar los malos tiempos de respuesta:

- Tiempos de respuesta lentos en transacción CICS
- Encolamiento de hilos trabajadores en el Gateway daemon
- Errores de Entrada/Salida durante las conexiones con el Gateway daemon
- Restricciones en la red entre el cliente remoto y el Gateway daemon
- Restricciones en la red entre el CTG y el CICS
- Stress en la JVM causando mal rendimiento en el Gateway daemon

Tiempos de respuesta lentos en transacción CICS

Tiempos lentos de procesamiento de las transacciones en el CICS causa incrementos en los tiempos de respuesta.

- **Escenario:** Esto puede ocurrir cuando el sistema CICS tiene restricciones o cuando los sistemas de bases de datos interconectados causan retrasos en el procesamiento de las transacciones.
- **Estadísticas a considerar:** CS_IAVRESP, CSx_IAVRESP
  - Si el valor de CS_IAVRESP es mayor que el que esperábamos para la transacción, el sistema CICS puede estar restringido o los sistemas de bases de datos interconectados pueden estar causando retrasos en el procesamiento de la transacción. Si un sistema CICS en particular está experimentando bajos tiempos de respuesta, hay que comprobar el valor de CSx_IAVRESP.
- **Solución:** Investigar los tiempos de respuesta de CICS usando las utilidades de monitorización del CICS y resolver las restricciones que se encuentren.

Encolamiento de hilos trabajadores en el Gateway daemon

Encolamiento de peticiones de transacciones en el CTG debido a un alto uso de hilos trabajadores.

- **Escenario:** Esto puede ocurrir cuando el número de administradores de conexiones asignados es mayor que el número de hilos trabajadores disponibles.
- **Estadísticas a considerar:** WT_ITIMEOUTS, WT_CCURR, WT_IALLOCHI
  1. Para saber si hay hilos trabajadores enconlados hay que considerar el valor de WT_ITIMEOUTS. Si el valor de WT_ITIMEOUTS > 0 entonces hay hilos trabajadores encolados.
  2. Si WT_CCURR = WT_IALLOCHI entonces todos los hilos trabajadores están en uso
- **Solución:** Incrementar el número de hilos trabajadores modificando el valor del parámetro *maxworker* para que sea igual al número de administradores de conexiones. Hay que considerar el reducir el valor del parámetro de configuración *workertimeout* si los tiempos de encolamiento son inaceptablemente altos.

Errores de Entrada/Salida durante las conexiones con el Gateway daemon

Un número insuficiente de administradores de conexiones está configurado en el CTG lo que causa errores de Entrada/Salida en el cliente remoto.

- **Escenario:** Esto ocurre cuando todos los hilos administradores de conexiones están asignados a clientes remotos.
- **Estadísticas a considerar:** CM_IALLOCHI, CM_SMAX
  - Si CM_IALLOCHI = CM_SMAX entonces todos los hilos administradores de conexiones están asignados a clientes remotos.
- **Solución:** Incrementar el número máximo de administradores de conexiones incrementando el valor del parámetro de configuración *maxconnect*. Considerar el número máximo de hilos trabajadores que hayamos definido.

Restricciones en la red entre el cliente remoto y el Gateway daemon

La transmisión de grandes cantidades de datos causa aumentos en los tiempos de respuesta debido al retardo en el envío de los datos con el Gateway daemon.

- **Escenario:** Esto ocurre cuando se transmiten grandes cantidades de datos, como COMMAREAs de 32KB, aunque no estemos usando toda la COMMAREA.
- **Estadísticas a considerar:** GD_LAVRESP, GD_IAVRESP, GD_IREQDATA, GD_IRESPDATA
    - Si los tiempos de respuesta en el cliente remoto son mayores que el valor reportado en las estadísticas GD_LAVRESP o GD_IAVRESP, esto puede ser debido a restricciones en la red entre el cliente remoto y el Gatewy daemon
- **Solución:** Seleccionar una de las siguientes:
    - Investigar y modificar, si es posible, el ancho de banda de la red.
    - Modificar el diseño de la aplicación para optimizar el envío de datos usando truncamiento de nulos en la COMMAREA o usando el método setCommareaOutboundLength() o setCommareaInboundLength().
    - Si las aplicaciones usan Contenedores, modificar el diseño de la aplicación para que use Contenedores más pequeños.

Restricciones en la red entre el CTG y el CICS

La transmisión de grandes cantidades de datos causa aumentos en los tiempos de respuesta debido a la latencia de la red en la conexión entre el CTG y el CICS.

- **Escenario:** Esto ocurre cuando se transmiten grandes cantidades de datos, como COMMAREAs de 32KB, aunque no estemos usando toda la COMMAREA.
- **Estadísticas a considerar:** GD_IAVRESP, CS_IAVRESP
    - Si GD_IAVRESP – CS_IAVRESP = “un valor alto” entonces puede haber restricciones en la red entre el CTG y el CICS.
- **Solución:** Seleccionar una de las siguientes:
    - Investigar y modificar, si es posible, el ancho de banda de la red.
    - Modificar el diseño de la aplicación para optimizar el envío de datos usando truncamiento de nulos en la COMMAREA o usando el método setCommareaOutboundLength() o setCommareaInboundLength().
    - Si estamos usando SNA sobre TCP/IP, considerar una solución de red que sólo use TCP/IP.

Stress en la JVM causando mal rendimiento en el CTG

En algunas circunstancias, el CTG puede sufrir un mal rendimiento debido a que gasta largos periodos asignando almacenamiento o realizando tareas recolección de basura o Garbage Collector de Java.

- **Escenario:** Esto puede ocurrir si el tamaño de Java heap se usa en un entorno donde se usan grandes cantidades de datos y un gran número de hilos trabajadores, más de 200, se están usando.
- **Estadísticas a considerar:** GD_IAVRESP, CS_IAVRESP, CM_IALLOCHI, WT_IALLOCHI, SE_CHEAPGCMIN, SE_SHEAPMAX, SE_IGCTIME, GD_IRUNTIME, SE_IGCCOUNT
    1. Tiempo de procesamiento en el CTG alto
        - Si GD_IAVRESP – CS_IAVRESP > 100 milisegundos entonces el tiempo de procesamiento del CTG es alto
    2. Se están encolando los administradores de conexiones debido a los hilos trabajadores

- Si (CM_IALLOCHI > WT_IALLOCHI) y WT_IALLOCHI > 0 entonces los administradores de conexiones se están encolando debido a hilos trabajadores.

3. Si ocurre alguna de las siguientes causas entonces el Garbage Collector de Java, GC, está sobrecargado:
   - GC no libera al menos el 50% del heap, esto es SE_CHEAPGCMIN/SE_SHEAPMAX > 50%
   - El tiempo gastado en GC es más del 10% del tiempo de procesamiento, esto es SE_IGCTIME*1000/GD_IRUNTIME > 10%
   - El periodo entre eventos del GC es menor de 1 por segundo, esto es GD_IRUNTIME/SE_IGCCOUNT < 1segundo

- **Solución:** Aumentar el tamaño mínimo y máximo del JVM heap y el tamaño de la región asociada al CTG.

2.9.3.- Retraso introducido por el CTG

Se han hecho varias mediciones para comprobar el retraso que añade el CTG a la hora de procesar las peticiones.

Por un lado se ha comprobado por medio de las estadísticas GD_LAVRESP y GD_IAVRESP los tiempos de respuesta que nos ofrece para el CTG.

Por otro lado, se ha comprobado el tiempo que transcurre desde que se envía una petición al CTG hasta que se obtiene respuesta. Después, por medio de un monitor de rendimiento que tiene el CICS, comprobamos el tiempo que ha costado ejecutar el programa que hemos pedido. Haciendo la resta entre ambos tiempos obtenemos el tiempo que ha transcurrido el CTG procesando la transacción.

En ambos casos, el tiempo de procesado del CTG es de aproximadamente 0.1-0.2 segundos. Teniendo en cuenta que los programas CICS que ejecutamos tardan en completarse aproximadamente 0.5 segundos creemos que el retardo añadido por el CTG no es considerablemente grande.

3.- CICS Service Flow Feature

3.1.- Introducción a SFF

El CICS Service Flow Feature, CICS SFF, CSFF o SFF que es como lo llamaremos a partir de ahora, es un añadido del CICS Transaction Server a partir de la versión 3. Permite agregar múltiples llamadas y automatizar interacciones con pantallas 3270 en lo que se llama un Service Flow, Flujo de Servicio o directamente flujo.

De esta forma podemos grabar flujos que simulen la interacción con una transacción conversacional o pseudo-conversacional, instalarlo en el CICS y luego invocarlo por medio del CICS Transaction Gateway. Esto nos permite llamar por medio del CTG a transacciones conversacionales o pseudo-conversacionales, cuya interacción con el usuario hemos grabado previamente, que es la principal característica que queremos explotar. Simplemente tenemos que invocar al flujo grabado pasándole los datos de entrada adecuados y el se preocupa de interactuar con las transacciones hasta devolvernos los datos de salida que hayamos definido.

También mejora la eficiencia al juntar múltiples peticiones en una sola y automatizar ciertas interacciones con la transacción.

Nos permite además invocar los flujos grabados desde servicios Web. Explicaremos como hacer esto aunque no es la principal característica que queremos explotar del SFF.

En la Figura 3.1 podemos ver un ejemplo de lo que sería un flujo. Aunque también permite llamar a aplicaciones basadas en COMMAREA, nosotros lo vamos a utilizar para interactuar con transacciones basadas en terminal 3270.

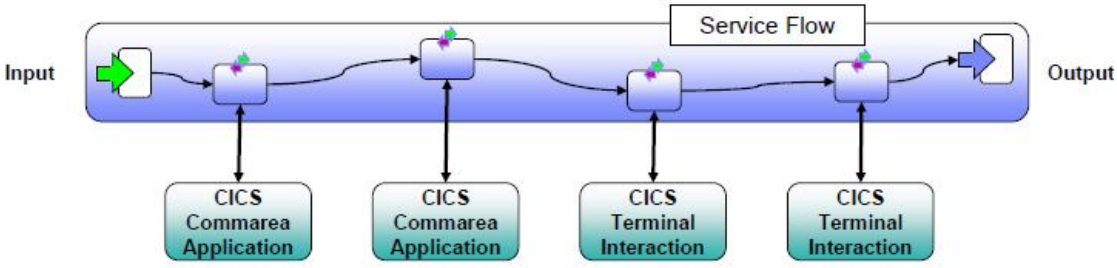


Figura 3.1 Flujo de SFF

El SFF consta de dos componentes: el SFR y el SFM.

El SFR o Service Flow Runtime es un componente que se instala en el mainframe y que permite ejecutar los flujos previamente grabados. Este componente es gratuito y lo podemos instalar en cualquier mainframe en el que tengamos una licencia del CICS TS.

El SFM o Service Flow Modeler es una herramienta gráfica que utilizamos para grabar los flujos y que se utiliza desde un ordenador personal. También permite transformar estos flujos en los componentes necesarios que podemos desplegar en el CICS para que funcionen los flujos grabados. El SFM forma parte del módulo Enterprise Service Tools que a su vez forma parte del Rational Developer for System z, o RDz. Para poder ejecutar este último necesitamos el Rational Business Developer. Estos programas son de pago y permiten realizar otras muchas tareas como desarrollar programas COBOL para CICS, crear servicios Web que podemos ejecutar en el mainframe, etc.

Por medio de estos dos componentes obtenemos lo siguiente:

- Un entorno gráfico de desarrollo que permite crear flujos a partir de interacciones con aplicaciones CICS.
- Un generador que permite transformar estos flujos en aplicaciones CICS que podemos ejecutar en el CICS TS.
- Un entorno de ejecución que permite ejecutar estos flujos como programas basados en Canales y Contenedores o COMMAREA y comos servicios Web.

Como ya hemos visto las ventajas que presentan los Canales y Contenedores frente a las COMMAREAS, vamos a hacer uso de esta primera opción para invocar los flujos que creemos. Esto nos limita a usar aplicaciones Java si utilizamos el CTG versión 7.2 aunque consideramos que no es una restricción demasiado importante.

También veremos como hacer uso de un servicio Web desde una aplicación Java o .Net.

Aunque en un principio pueda parecer un gran inconveniente tener que crear un flujo por cada tarea que queramos realizar con una transacción, la propia experiencia nos dice que grabar un flujo con 5 o 6 interacciones con pantallas no cuesta más de 15 minutos. Este tiempo es bastante menor al que costaría adaptar los programas basados en terminal para que puedan funcionar como programas basados en COMMAREA o en Canales y Contenedores. Además, este tiempo a la larga lo vamos a recuperar si hacemos bastante uso de las mismas tareas al no tener que volver a introducir los mismos datos y opciones de navegación entre pantallas una y otra vez. Ahora simplemente tenemos que introducir los datos de entrada que cambien en cada ejecución y el SFF se encarga de hacer las interacciones con las pantallas.

Por otra parte, al poder hacer uso ahora del CTG para llamar a los flujos, podemos modernizar las pantallas tanto de entrada como de salida de datos para hacerlas visualmente más agradables y poder hacer uso de todas las características que nos ofrece una interfaz gráfica moderna.

3.2.- Instalación de SFR y SFM

Como hemos dicho, el SFF consta de los componentes SFM y SFR. Para poder hacer uso del SFF necesitamos instalar ambos componentes.

3.2.1.- Instalación del SFM

La instalación del SFM en la máquina cliente que vamos a utilizar para grabar los flujos no presenta mayores dificultades. Es similar a la instalación de la mayoría de programas en un

entorno Windows. Simplemente tenemos que instalar primero el Rational Business Developer y a continuación el Rational Developer for System z asegurándonos que también se instala el módulo Enterprise Service Tools. Para hacer esto simplemente tenemos que introducir el CD o DVD que contiene el programa y seguir las instrucciones que aparecen por pantalla.

3.2.2.- Requisitos de instalación del SFR

Antes de empezar a instalar el SFR tenemos que comprobar los requisitos que presenta para la versión que queremos instalar. Nosotros vamos a instalar el SFR versión 3.2 y sus requisitos son los siguientes:

- CICS Transaction Server v3.2
- Enterprise COBOL para z/OS v3.2 o superior
  - Esto es debido a que los programas que se generan a partir de los flujos están escritos en COBOL
- z/OS Language Environment v1.7 o superior

3.2.3.- Instalación del SFR

Una vez que hemos comprobado que nuestro sistema cumple los requisitos para poder instalar el SFR, vamos a ver que pasos hay que realizar para instalarlo en el CICS1 para el que hemos configurado anteriormente acceso al CTG.

Al igual que con el CTG, la instalación del SFR en el mainframe consta de dos partes.

La primera de ellas consiste en volcar los data sets y miembros necesarios para la instalación y funcionamiento del SFR desde la cinta donde IBM suministra sus productos hasta el mainframe. Al igual que con el CTG, esta parte ya estaba realizada al empezar a hacer el proyecto por lo que no ha sido posible documentarla. Los pasos necesarios para llevar a cabo este proceso vienen detallados en el GI13-0525-00 - CICS Service Flow Feature for Transaction Server for z/OS V3.2 Program Directory. En esta parte le decimos el data set donde queremos que instale el SFF que ha sido el CICS.V3R2M0.SFF

La segunda parte consiste en configurar los JCLs DFHMAINJ y DFHMASET según nuestra instalación y ejecutarlos. Ambos JCLs se encuentran en el data set CICS.V3R2M0.SFF.SCIZSAMP donde almacenamos los miembros sin modificar para poder instalar el SFF en más de un CICS. Este data set se ha especificado en la primera parte de la instalación. Estos JCLs van a realizar las siguientes funciones:

- DFHMAINJ
  - Crea los data sets donde se van a dejar los módulos configurados para nuestra instalación
- DFHMASET
  - Compila los programas necesarios para que funcione el SFR, crea e inicializa los ficheros necesarios para que funcione y define los recursos necesarios en el CICS

3.2.3.1.- Configuración y ejecución del DFHMAINJ

Como ya hemos dicho, este JCL crea los data sets y miembros configurados para nuestra instalación. En especial crea los siguientes data sets:

- SCIZSAMP

    - Crea los miembros que había en el data set de instalación pero configurados para nuestra instalación particular
- SCIZLOAD
    - Contiene copybooks, trozos de código que van a usar otros miembros y programas para realizar las tareas adecuadas
- SCIZMAC
    - Contiene los ejecutables compilados de los programas del SFR

Las variables que utiliza este JCL y los valores que le damos para que se adecue a nuestra instalación son los siguientes:

- SHLQ : CICS.V3R2M0.SFF
    - Data set donde hemos instalado el SFF en la primera parte
- QUAL : CICS1.V3R2M0.SFF
    - Data set donde dejamos los miembros configurados para esta instalación
- VOLSER : SSS111
    - Nombre del volumen de almacenamiento donde vamos a guardar los data sets
- RDOLIST : CICSSFR
    - Nombre del grupo CICS donde van a quedar las definiciones de los recursos
- CSDNAME : CICS1.V3R2M0.DFHCSD
    - Nombre del CSD donde vamos a tener las definiciones de los recursos. Es un archivo donde el z/OS almacena las definiciones de recursos para cada CICS
- HLQCICS : CICS.V3R2M0.CICS
    - Cualificador donde están los data sets del CICS
- HLQCOBOL : IGY.Z180
    - Cualificador donde están los data sets de ejecución de COBOL
- HLQCEE : CEE.Z180
    - Cualificador donde están los data sets de ejecución del Language Environment
- PREFIX : SFFR
    - Prefijo que se le da a algunos archivos que se crean

A continuación vamos a ver las variables que necesitamos para configurar correctamente la opción de invocar flujos desde servicios Web:

- WSDIR_REQ : /wsbind/requester/
    - Directorio donde se dejarán los servicios Web
- CONFIG_REQ : /usr/lpp/cics/cicsts32/samples/pipelines/basicsoap11requester.xml
    - Archivo de configuración de la pipeline en modo solicitante
- SHELF_REQ : /var/cics/requester/
    - Directorio que contendrá los subdirectorios de la pipeline en modo solicitante
- WSDIR_PROV : /wsbind/provider/
    - Directorio que contendrá el archivo Web Service Binding

  CONFIG_PROV : /usr/lpp/cics/cicsts32/samples/pipelines/basicsoap11provider.xml
    - Archivo de configuración de la pipeline en modo proveedor
- SHELF_PROV : /var/cics/provider/
    - Directorio que contendrá los subdirectorios de la pipeline en modo proveedor

Para abrir este JCL en edición seguimos los pasos que hemos visto en el apartado 2.3.2.3 de instalación del CTG para abrir un miembro de un data set en edición.

En el Documento 3.1 vemos un ejemplo de cómo quedaría el JCL DFHMAINJ configurado para nuestra instalación y vamos a intentar explicar a grandes rasgos que es lo que hace. Las líneas que empiezan por //* son comentarios.

```
//DFHMAINJ JOB (DI770SE04),'USER',
//            CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
//            NOTIFY=&SYSUID
//
//* Si fuésemos a submitir el DFHMAINJ más de una vez, a partir de la segunda
//* ejecución comentamos las siguientes 8 línea para que borrase los data
//* sets que ya habríamos creado con la primera ejecución del JCL
//*SYSD2     EXEC PGM=IKJEFT01
//*SYSUDUMP  DD SYSOUT=*
//*SYSTSPRT  DD SYSOUT=*
//*SYSPRINT  DD SYSOUT=*
//*SYSTSIN   DD *
//* DEL 'CICS1.V3R2M0.SFF.SCIZSAMP'
//* DEL 'CICS1.V3R2M0.SFF.SCIZLOAD'
//* DEL 'CICS1.V3R2M0.SFF.SCIZMAC'
//*
//*
//* Crea los data sets que hemos indicado e invoca el JCL DFHMAINR que
//* configura los miembros del SCIZSAMP según nuestra instalación
//BORRAR  EXEC PGM=IEFBR14
//UNOEL   DD DSN=CICS1.V3R2M0.SFF.SCIZSAMP,DISP=(SHR,DELETE,DELETE)
//DOSEL   DD DSN=CICS1.V3R2M0.SFF.SCIZLOAD,DISP=(SHR,DELETE,DELETE)
//TRESL   DD DSN=CICS1.V3R2M0.SFF.SCIZMAC,DISP=(SHR,DELETE,DELETE)
//*
//CREATE   EXEC PGM=IEFBR14
//SYSPRINT DD  DUMMY
//SYSIN    DD  DUMMY
//SCIZSAMP DD DSN=CICS1.V3R2M0.SFF.SCIZSAMP,
//             DISP=(NEW,CATLG),
//             UNIT=SYSDA,
//             DSNTYPE=LIBRARY,
//             SPACE=(CYL,(10,5,25)),
//             DSORG=PO,RECFM=FB,LRECL=80,BLKSIZE=27920
//SCIZLOAD DD DSN=CICS1.V3R2M0.SFF.SCIZLOAD,
//             DISP=(NEW,CATLG),
//             UNIT=SYSDA,
//             DSNTYPE=PDS,
//             SPACE=(CYL,(2,2,10)),
//             DSORG=PO,RECFM=U,LRECL=0,BLKSIZE=32720
//SCIZMAC DD DSN=CICS1.V3R2M0.SFF.SCIZMAC,
//             DISP=(NEW,CATLG),
//             UNIT=SYSDA,
//             DSNTYPE=LIBRARY,
//             SPACE=(CYL,(1,1,10)),
//             DSORG=PO,RECFM=FB,LRECL=80,BLKSIZE=27920
//*
//COPY     EXEC PGM=IEBCOPY,REGION=8M
//SYSPRINT DD SYSOUT=*
//INL      DD DISP=SHR,
//     DSN=CICS.V3R2M0.SFF.SCIZLOAD
//INM      DD DISP=SHR,
//     DSN=CICS.V3R2M0.SFF.SCIZMAC
//OUTL     DD DISP=SHR,
//     DSN=CICS1.V3R2M0.SFF.SCIZLOAD
```

```
//OUTM     DD DISP=SHR,
//     DSN=CICS1.V3R2M0.SFF.SCIZMAC
//SYSIN    DD *
 COPYMOD INDD=((INL,R)),OUTDD=OUTL
 COPY    INDD=((INM,R)),OUTDD=OUTM
/*
//REXX     EXEC PGM=IKJEFT01,REGION=2M,DYNAMNBR=99
//SYSPROC DD DSN=CICSTS.V3R2M0.SFF.SCIZSAMP,
//             DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  %DFHMAINR validate
/*
//IN       DD *
/*
//* Definimos las variables necesarias con sus valores correspondientes
//* para que se configuren adecuandamente los miembros necesarios
JOB1        //++++++++ JOB ,CLASS=A,REGION=0M,
JOB2        //         NOTIFY=&SYSUID,MSGCLASS=X
JOB3        //*
JOB4        //* CLASS=M y MSGCLASS=H por defecto
JOB5        //*
*
SHLQ        CICS.V3R2M0.SFF
QUAL        CICS1.V3R2M0.SFF
VOLSER      SSS111
RDOLIST     CICSSFR
CSDNAME     CICS1.V3R2M0.DFHCSD
HLQCICS     CICS.V3R2M0.CICS
HLQCOBOL    IGY.Z180
HLQCEE      CEE.Z180
WSDIR_REQ   /wsbind/requester/
CONFIG_REQ  /usr/lpp/cicsts/cicsts32/samples/pipelines/\
            \basicsoap11requester.xml
SHELF_REQ   /var/cicsts/requester/
WSDIR_PROV  /wsbind/provider/
CONFIG_PROV /usr/lpp/cicsts/cicsts32/samples/pipelines/\
            \basicsoap11provider.xml
SHELF_PROV  /var/cicsts/provider/
PREFIX      SFFR
*
/*
//
```

Documento 3.1 JCL DFHMAINJ configurado para nuestra instalación

Una vez que hemos configurado correctamente el JCL, en la parte superior del la pantalla tenemos una línea de comandos donde escribimos el “sub” y pulsamos Control para submitir el JCL como podemos ver en la Figura 3.2.

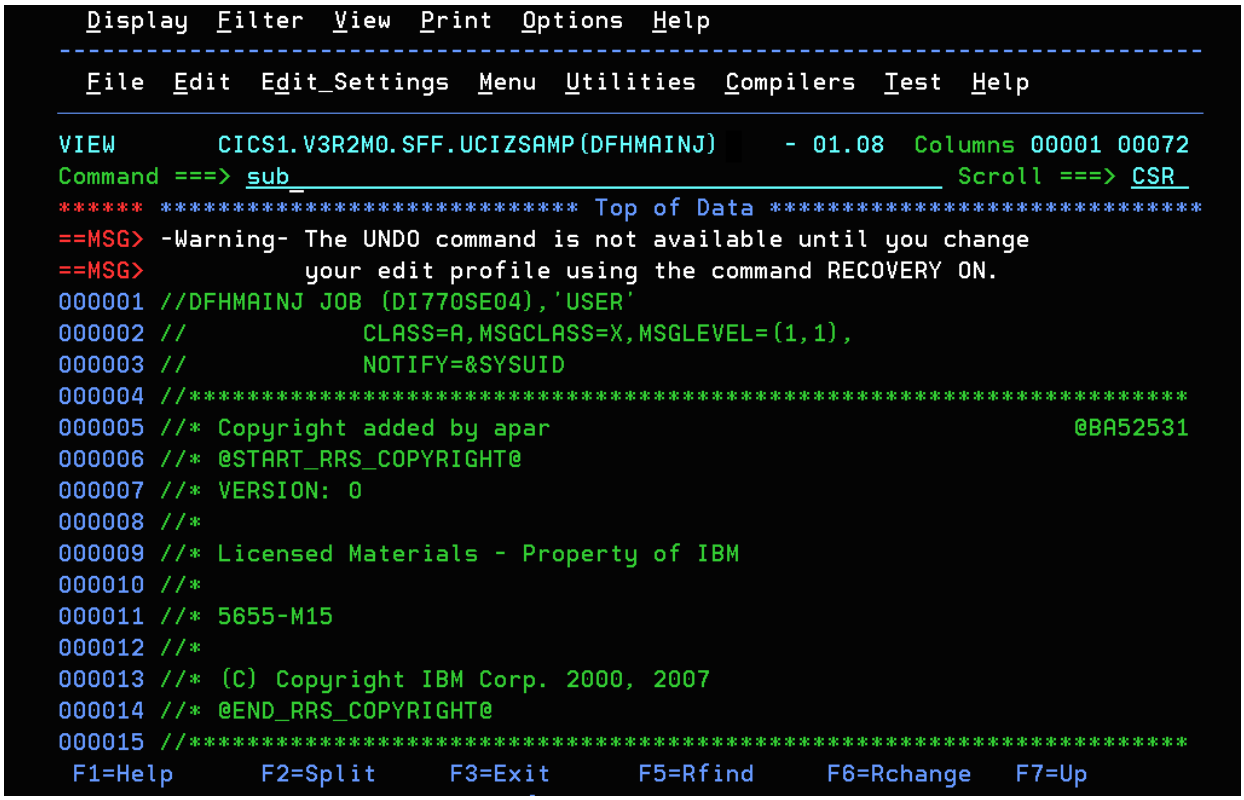


Figura 3.2 JCL DFHMAINJ listo para ser submitido

Cuando el JCL finalice su ejecución nos aparecerá el mensaje que podemos ver en la Figura 3.3 donde el RC=00 nos indica que no se han producido errores ejecutando el JCL.

Figura 3.3 Mensaje de finalización correcta de JCL

En la cola de salida del JCL aparecerán los siguientes mensajes que indican que el JCL se ha ejecutado correctamente:

- DFHMAI1002I SCIZSAMP customization beginning.
- DFHMAI1000I Validation of input parameters is taking place.
- DFHMAI1011I SCIZSAMP customization ended without errors.

3.2.3.2.-Configuración y ejecución del DFHMASET

Este JCL ha quedado configurado al ejecutar el DFHMAINJ por lo que las variables que utiliza ya tienen los valores adecuados. Simplemente tenemos que abrirlo, ya sea en modo visualización o edición, para comprobar que los valores de las variables son adecuados y submitirlo. Cuando finalice nos aparecerá un mensaje similar al que hemos visto al submitir el DFHMAINJ.

Una vez que hemos ejecutado el DFHMASET, tenemos que modificar la Sysin del CICS donde queremos instalar el SFR que en este caso es el CICS1. La Sysin es un miembro de configuración del CICS que permite configurar ciertos parámetros. En nuestro caso se encuentre en el data set CICS1.USER.PARMLIB. Añadimos la siguiente línea a lo que ya contenga: INITPARM=(DFHMAINS='/repositorio/sff/')

En este directorio de la parte Unix del z/OS es donde vamos a almacenar los archivos spf. Estos son archivos de propiedades de los flujos que generemos con el SFM. En este directorio es donde va a buscar estos archivos la transacción CMAN, que gestiona los flujos, para instalarlos.

Una vez que hagamos esto tenemos que parar y arrancar el CICS para que recoja los cambios y ya tenemos todo listo para poder usar el SFF.

3.3.- Flujo básico e instalación de flujos

3.3.1.- Crear el entorno de trabajo y configurarlo

Una vez que tenemos instalado y funcionando el SFF, vamos a ver como crear un flujo básico, generar los componentes necesarios y desplegarlos e instalarlo en el CICS.

Lo primero que tenemos que hacer es abrir el Rational Developer for System z, que a partir de ahora llamaremos RDz. Nos aparecerá la ventana de la Figura 3.4 donde podemos elegir el espacio de trabajo que queremos entre los existentes o crear uno nuevo.

Figura 3.4 Elección del espacio de trabajo

El espacio de trabajo es el directorio donde se van a guardar todos los flujos y archivos que creemos.

Una vez que hayamos elegido, o creado, nuestro espacio de trabajo pulsamos “Aceptar” y nos encontraremos con la pantalla de la Figura 3.5. Esta es una pantalla de bienvenida con links a algunos tutoriales que nos permiten empezar a trabajar con RDz aunque ninguno nos sirve para las tareas que vamos a realizar.

Esta pantalla aparecerá cada vez que creemos un nuevo espacio de trabajo.

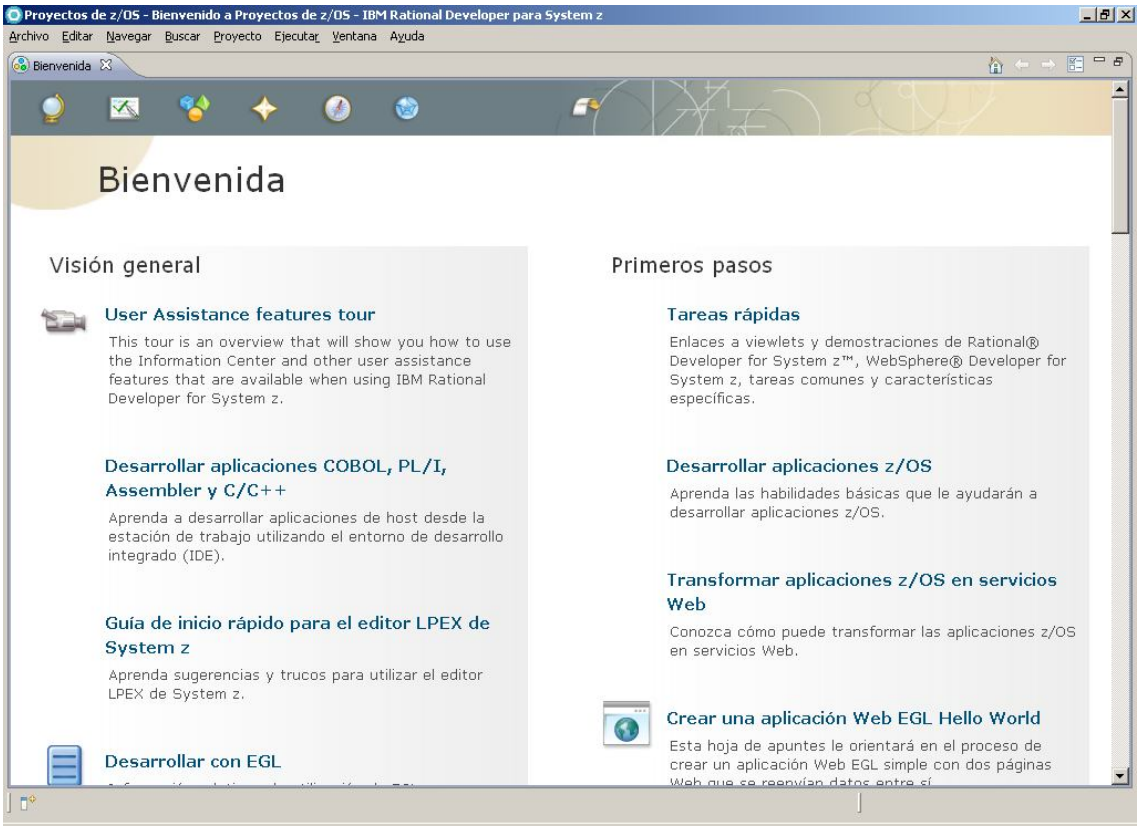


Figura 3.5 Pantalla de bienvenida del RDz

Cerramos la pestaña de “Bienvenida” pulsando la x de la parte superior. Esta pantalla ya no nos aparecerá las próximas veces que usemos este espacio de trabajo. Una vez cerrada la pantalla de bienvenida nos aparece la ventana que podemos ver en la Figura 3.6 que es la perspectiva de “Proyectos de z/OS”.

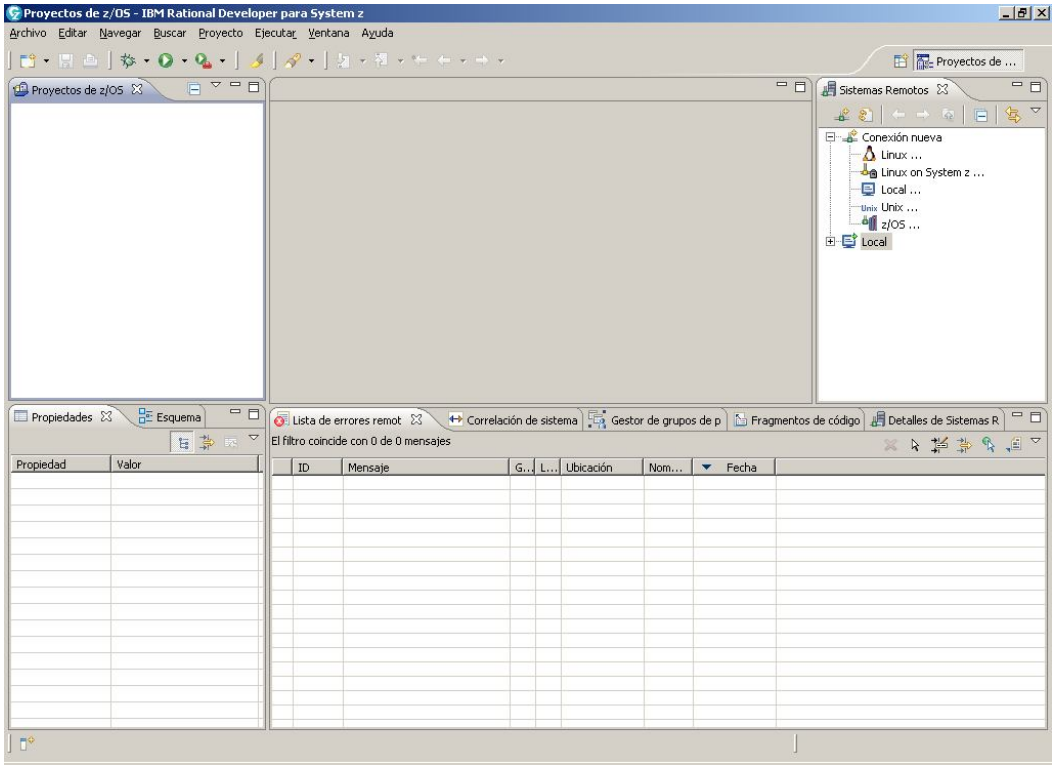


Figura 3.6 Perspectiva de “Proyectos de z/OS”

Esta perspectiva no es la que necesitamos para grabar flujos. Tenemos que cambiar a la perspectiva "Enterprise Service Tools". Para en la parte superior derecha de la pantalla pulsamos el icono  para que nos muestre otras perspectivas y seleccionamos "Enterprise Service Tools" como podemos ver en la Figura 3.7.

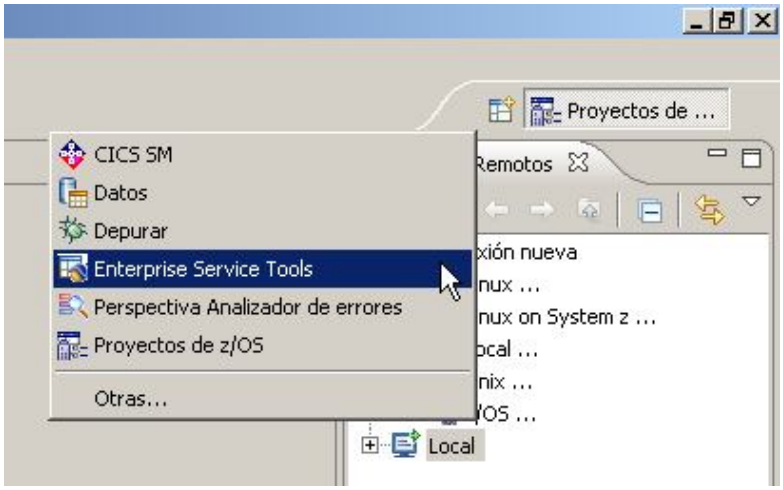


Figura 3.7 Selección de perspectiva "Enterprise Service Tools"

Con esto nos aparecerá la perspectiva de "Enterprise Service Tools" que podemos ver en la Figura 3.8.

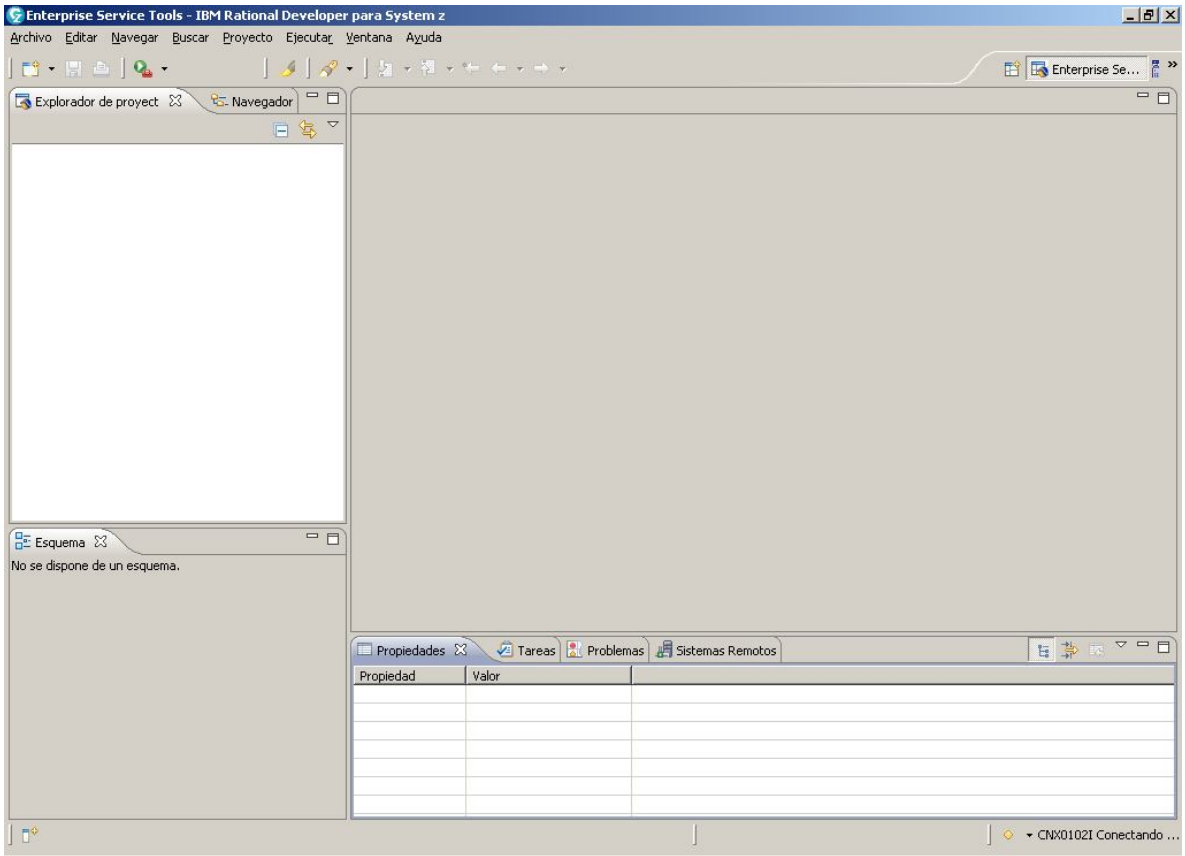


Figura 3.8 Perspectiva de Enterprise Service Tools

Esta es la perspectiva que vamos a utilizar para grabar y generar nuestros flujos. En esta perspectiva podemos ver en la parte superior izquierda un "Explorador de proyectos" donde veremos todos los archivos que se van creando conforme creemos nuevos proyectos y flujos. Los diferentes archivos que se van generando en esta parte los veremos conforme los vayamos necesitando y en un resumen al final de este capítulo. En la parte superior derecha que ahora está en gris se abrirán cualquier elemento que queramos abrir como conexiones, archivos para editarlos, etc. En la parte inferior izquierda tenemos el "Esquema" donde iremos viendo un esquema del elemento que tengamos abierto. En la parte inferior derecha tenemos diferentes pestañas, como "Propiedades", "Sistemas Remotos", etc. que nos van a permitir realizar varias tareas que iremos viendo conforme necesitemos.

Antes de empezar a grabar un flujo, necesitamos cambiar algunas de las preferencias que el RDz tiene por defecto. Antes de modificar preferencias, vamos a necesitar cargar unas plantillas de JCL que se encuentran en el mainframe por lo que vamos a crear una conexión al mainframe para poder importarlas. Para ello nos dirigimos a la pestaña de "Sistemas Remotos" que se encuentra en la parte inferior de la ventana como podemos ver en la Figura 3.9.

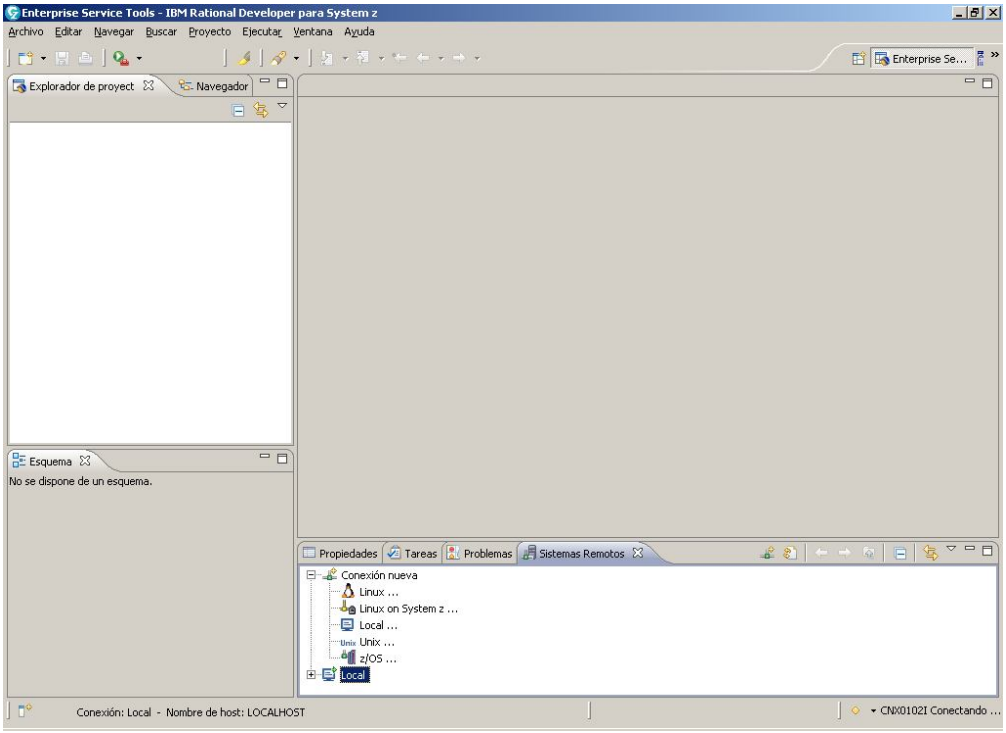


Figura 3.9 Pestaña de "Sistemas Remotos"

Una vez en esta pestaña hacemos clic derecho sobre "z/OS…", que es a donde queremos establecer la conexión, y seleccionamos "Conexión nueva…" como se muestra en la Figura 3.10.

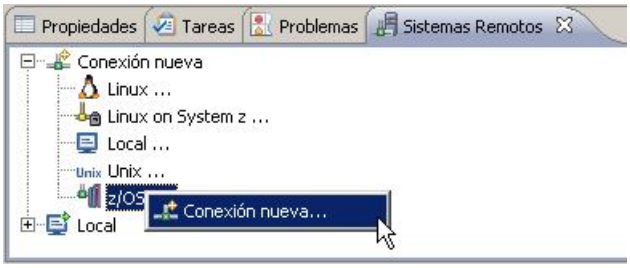


Figura 3.10 Selección de "Conexión nueva…"

Nos aparece la ventana de la Figura 3.11 que rellenamos con los datos adecuados como son la IP o nombre del mainframe al que nos queremos conectar y un nombre para esta conexión.

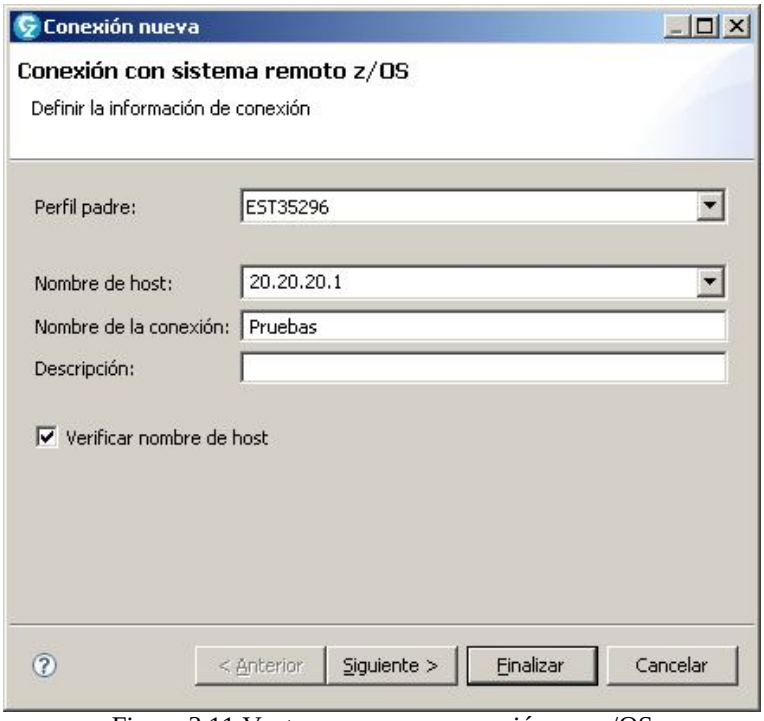


Figura 3.11 Ventana para crear conexión con z/OS

Nos aparece un nuevo nodo con el nombre “Pruebas” como podemos ver en la Figura 3.12.

Figura 3.12 Nodo “Pruebas” con la nueva conexión

Si intentamos desplegar los “Archivos MVS” que es donde están las plantillas nos aparecerá la ventana de la Figura 3.12 para que introduzcamos un usuario y contraseña de RACF válidos.

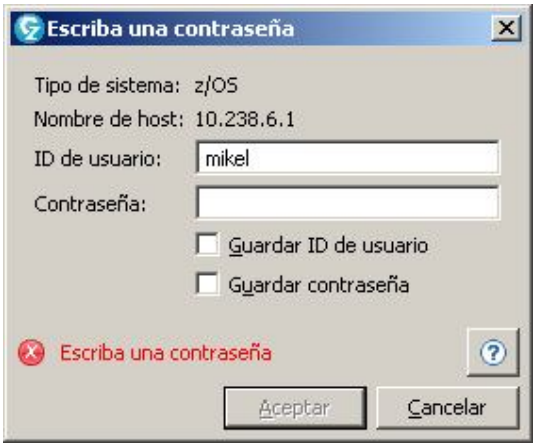


Figura 3.12 Ventana de identificación

Una vez introducidos los datos correctos, nos aparecerá una ventana indicándonos que la conexión no está protegida mediante SSL y a continuación los archivos MVS a los que tenemos acceso. Las plantillas que queremos importar no se encuentran en los archivos mostrados por lo que tenemos que crear un filtro que nos permita acceder a ellas. Para ello hacemos clic derecho sobre "Mis conjuntos de datos" y elegimos "Nuevo → Filtro…" como podemos ver en la Figura 3.13.

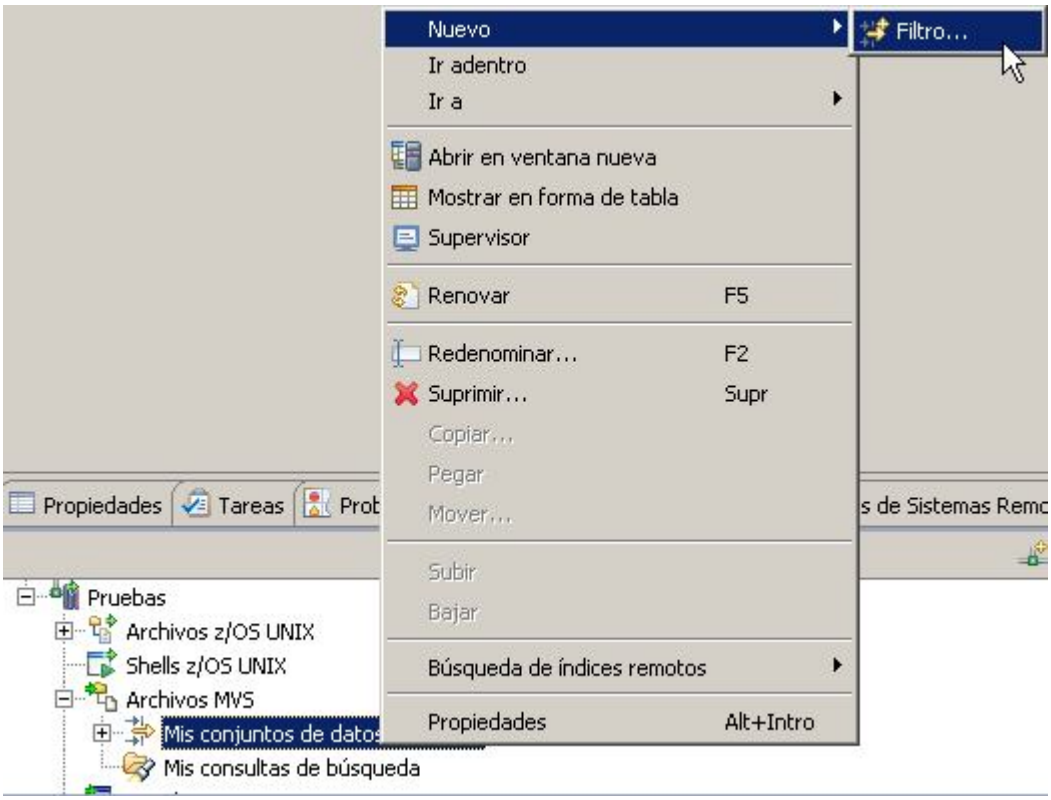


Figura 3.13 Selección de nuevo filtro de archivos MVS

Nos aparece la ventana de la Figura 3.14 donde introducimos el cualificador de o de los data sets que queremos mostrar.

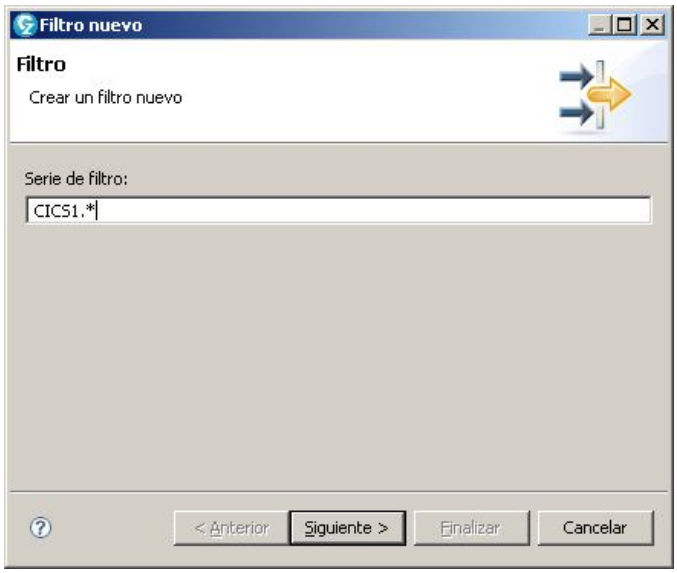


Figura 3.14 Nombre del data set a mostrar

Pulsamos “Siguiente” con lo que nos aparece la ventana de la Figura 3.15 para introducir el nombre del filtro y cuando lo hagamos pulsamos “Finalizar”.

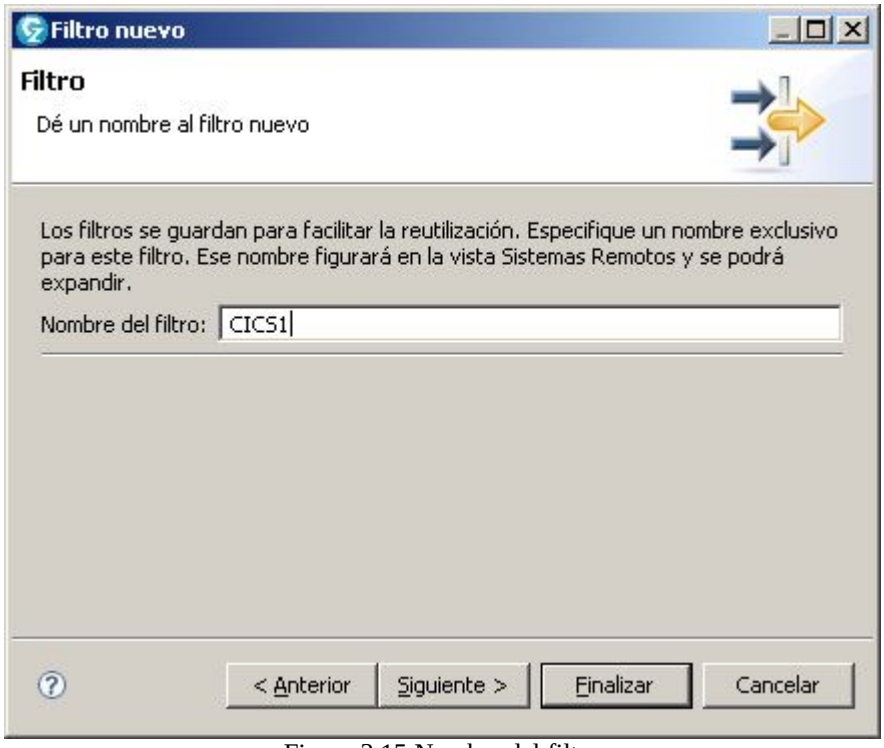


Figura 3.15 Nombre del filtro

Como podemos ver en la Figura 3.16 nos aparecerán varios data sets entre los que se encuentre el CICS1.V3R2M0.SFF.SCIZSAMP, donde están las plantillas. Si hemos modificado las plantillas que vienen por defecto con el SFF, que no es el caso, es probable que estas plantillas se encuentren en la librería UCIZSAMP en vez de en la SCIZSAMP.

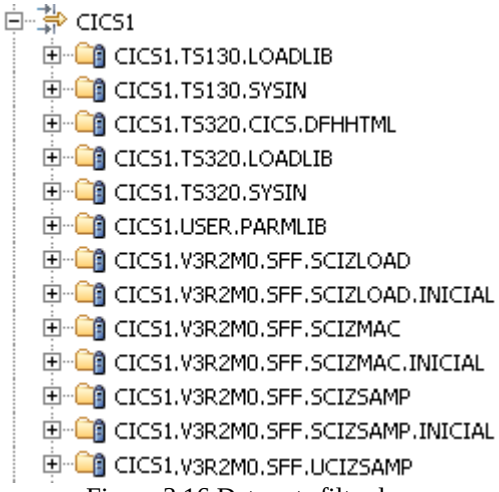


Figura 3.16 Data sets filtrados

Ahora ya tenemos todo lo necesario para modificar las preferencias. Para ello, en la barra de menús elegimos “Ventana → Preferencias”, con lo que se nos abre la ventana de la Figura 3.17.

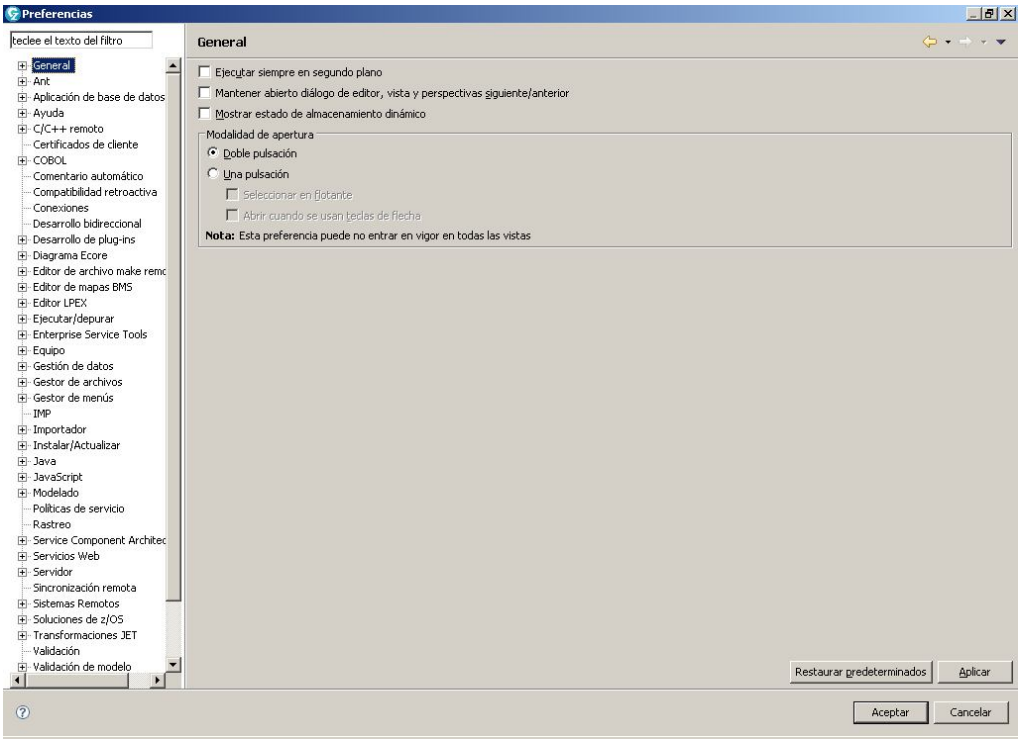


Figura 3.17 Preferencias de nuestro espacio de trabajo

En la parte superior izquierda vemos que tenemos un recuadro que usaremos para filtrar las preferencias que queremos buscar. Concretamente, vamos a modificar el valor de las páginas de códigos y las plantillas JCL. Para modificar el valor de la página de códigos, en el recuadro de filtro tecleamos “página” y seleccionamos la preferencia “Valores de página de códigos” que se ve en la Figura 3.18. Cambiamos el código de página por la “1145 España Euro” que es la que usa el mainframe. Esto es muy importante ya que sino tendremos problemas más adelante al subir los archivos al mainframe.

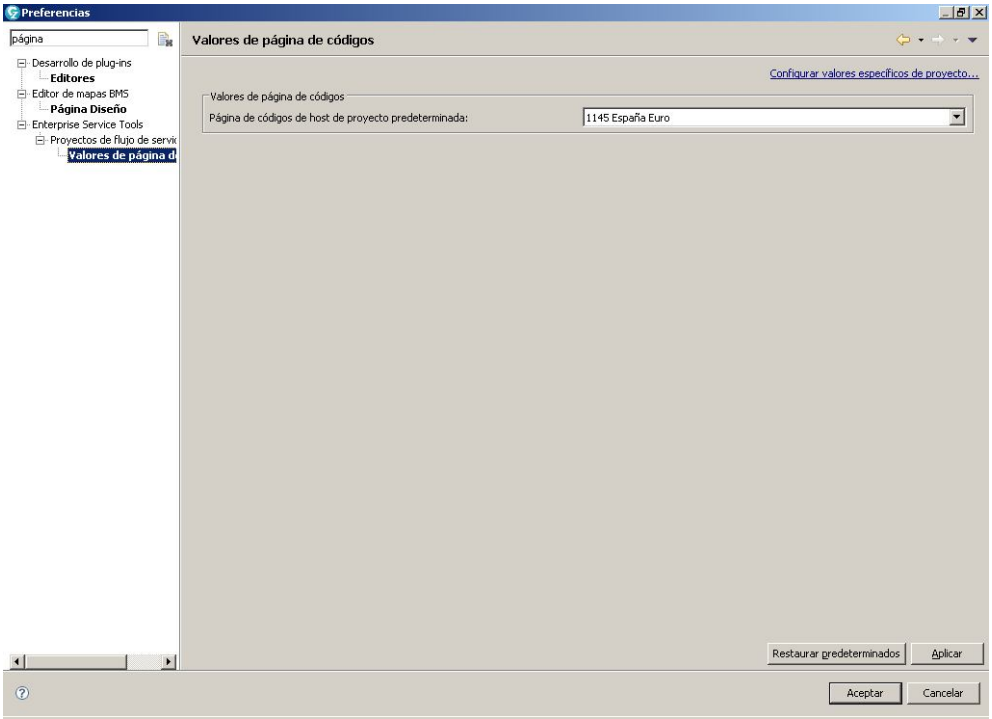


Figura 3.18 Valores de página de códigos

Para poder cambiar las plantillas JCL, tecleamos en el cuadro de filtro “jcl” y entre las opciones que nos aparecen elegimos “Plantillas JCL”. De esta forma nos aparecerá la ventana de la Figura 3.19.

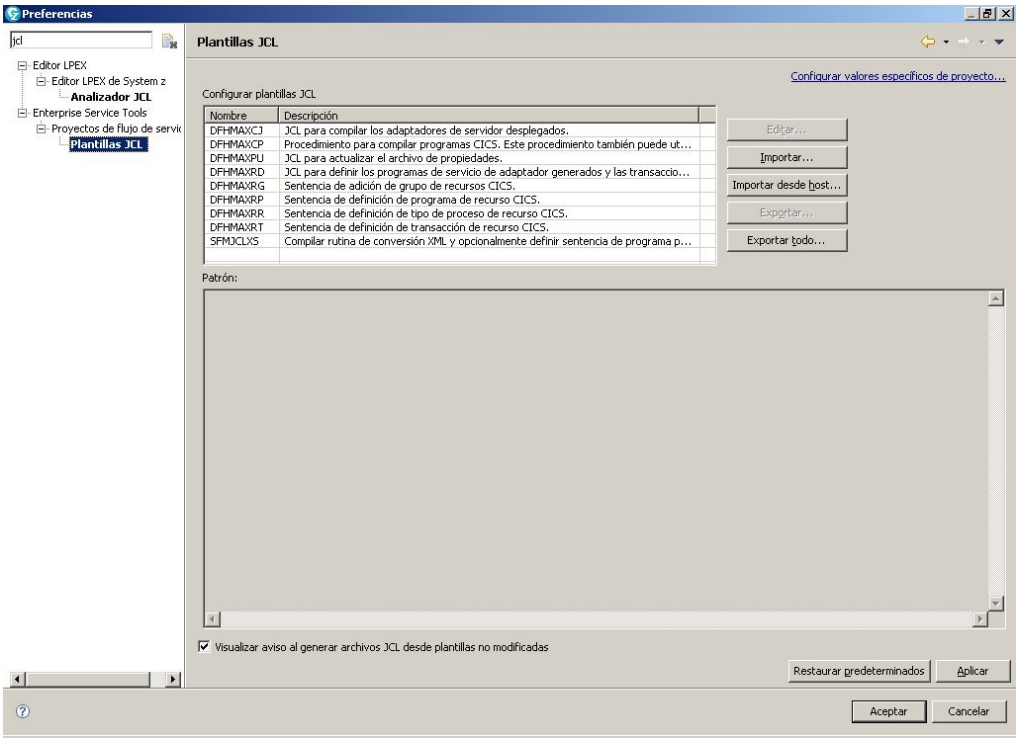


Figura 3.19 Plantillas JCL

Como no hemos transferido las plantillas a nuestra máquina y se encuentran en el mainframe hacemos clic en “Importar desde host…”. Con esto nos aparecerá la ventana de la Figura 3.20.

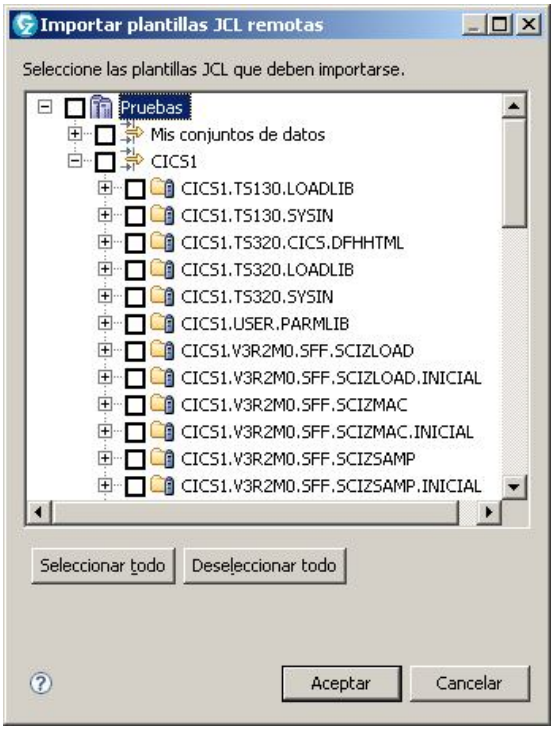


Figura 3.20 Ventana para importar plantillas desde el mainframe

Las plantillas que queremos importar se encuentran en el data set CICS1.V3R2M0.SFF.SCIZSAMP y son las siguientes:

- DFHMAXCJ
- DFHMAXCP
- DFHMAXRD
- DFHMAXRG
- DFHMAXRP
- DFHMAXRR
- DFHMAXRT

Una vez que las tenemos seleccionadas pulsamos "Aceptar" y tras unos momentos nos aparecerá la ventana de la Figura 3.21 indicándonos que las plantillas se han importado correctamente.

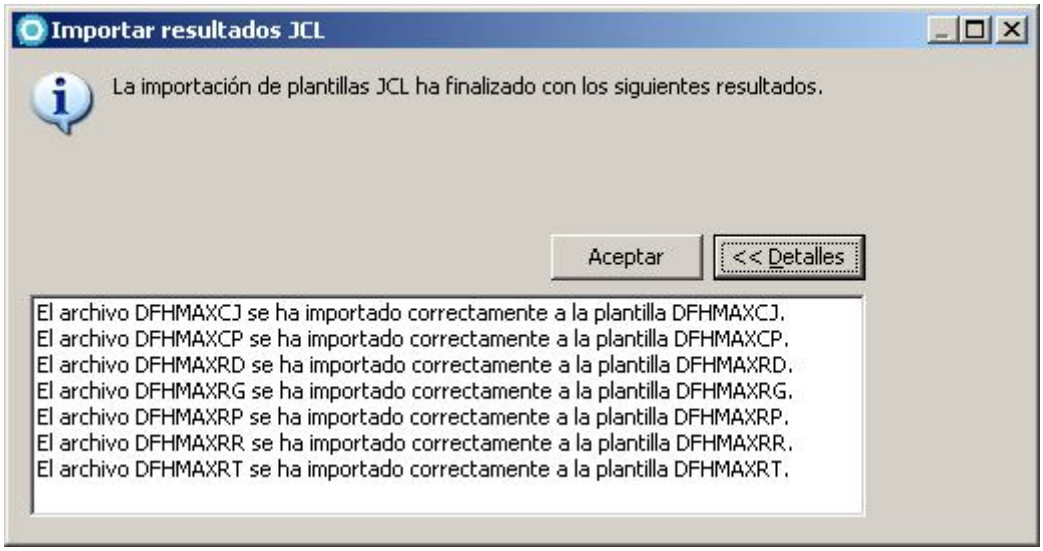


Figura 3.21 Importación correcta de plantillas JCL

Pulsamos "Aceptar" en esta ventana y en la ventana de preferencias. Con esto ya hemos modificado las preferencias necesarias y podemos empezar a grabar un flujo.

Estos pasos los tendremos que realizar cada vez que creemos un nuevo espacio de trabajo.

3.3.2.- Grabación de un primer flujo básico

Una vez que tenemos configurado el espacio de trabajo como queremos, vamos a grabar nuestro primer flujo. Para ello vamos a utilizar una transacción de ejemplo de IBM llamada EGUI que simula un catálogo de productos de una papelería.

El flujo que vamos a crear es muy básico. Simplemente tendrá dos variables de entrada para pedir los detalles de un producto y una de salida donde devolver los detalles de este producto. Más adelante veremos como grabar flujos más complejos con caminos alternativos y bucles.

Lo primero que tenemos que hacer antes de grabar un flujo es crear el proyecto que lo va a contener. Para ello hacemos clic derecho en la parte de "Explorador de proyectos" o haciendo

clic sobre el botón "Archivo" del menú superior y elegimos "Nuevo → Proyecto de flujo de servicios" como se puede ver en la Figura 3.22.

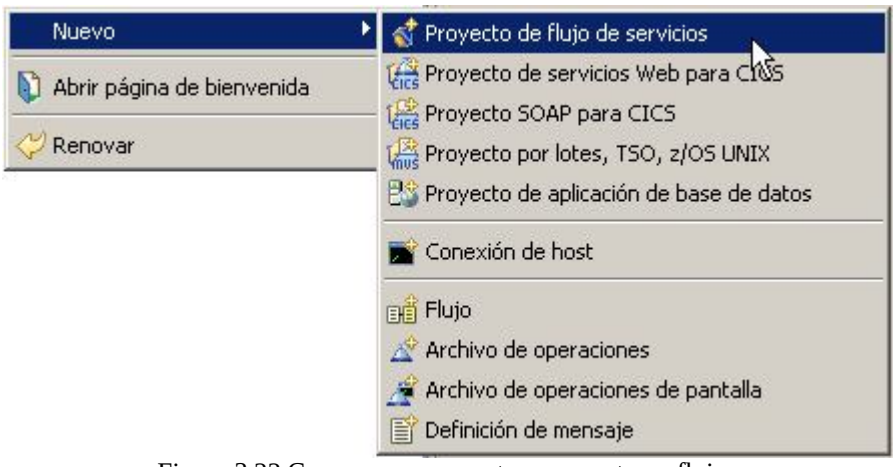


Figura 3.22 Crear nuevo proyecto para contener flujos

Aparece la pantalla de la Figura 3.23 donde le damos el nombre a nuestro proyecto, que en este caso será "pruebasEGUI" y pulsamos "Siguiente". Podríamos guardarlo en una ubicación diferente a donde se ha almacenado el espacio de trabajo pero no nos interesa.

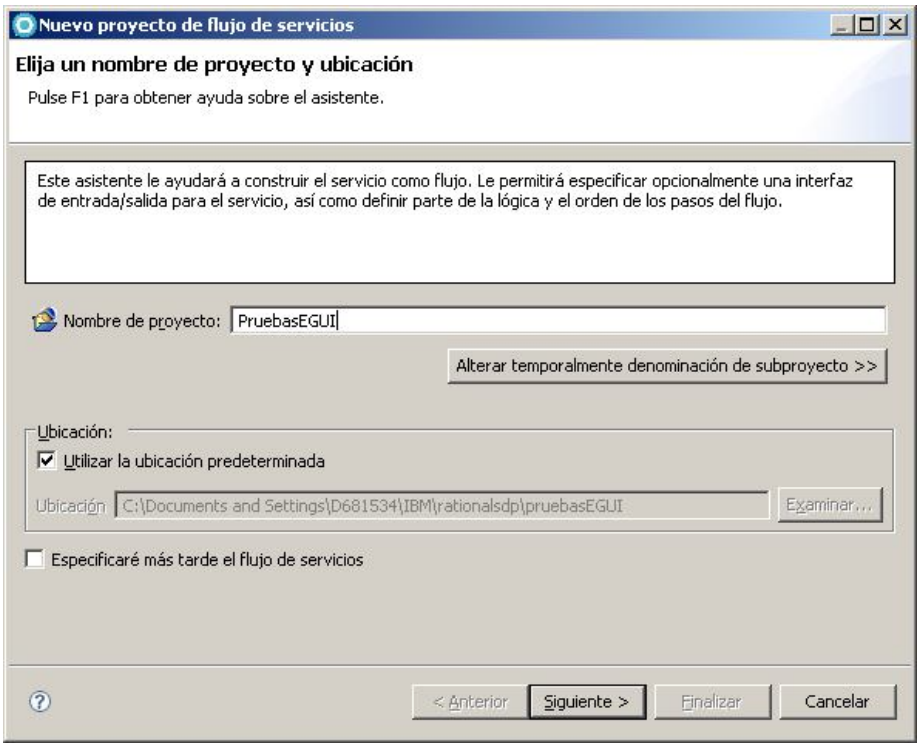


Figura 3.23 Nombre del proyecto

En la siguiente ventana, que podemos ver en la Figura 3.24, elegimos "Definir más tarde" y pulsamos "Siguiente".

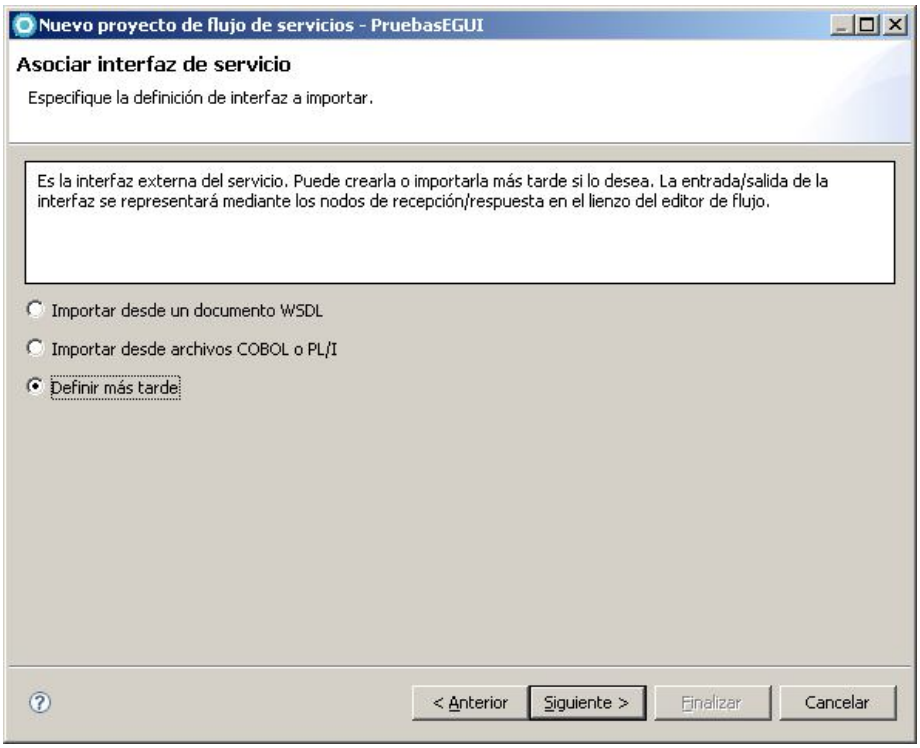


Figura 3.24 Definir interfaz de servicio

En la ventana de la Figura 3.25 podemos elegir el servicio que queremos crear. Como queremos grabar un flujo que simule una interacción con una aplicación de terminal elegimos "Registrar interacciones con una aplicación de terminal" y pulsamos "Siguiente".

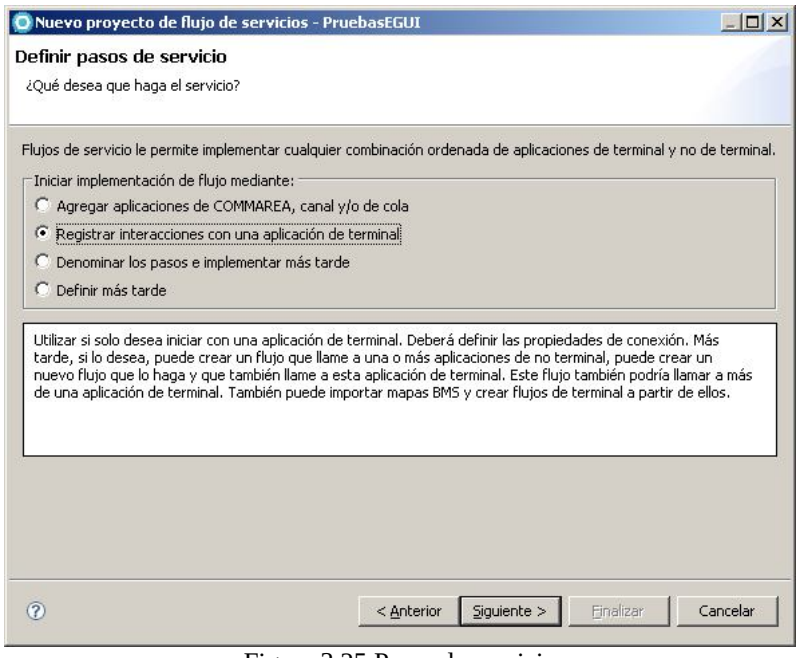


Figura 3.25 Pasos de servicio

En la última ventana vamos a configurar la conexión con el mainframe. Para ello le tenemos que dar un nombre a la conexión, que le hemos llamado "Pruebas" porque es a donde nos vamos a conectar, la IP donde está, que es la 20.20.20.1 y la página de códigos que como ya hemos dicho es my importante que sea la 1145. Si hemos modificado las preferencias como hemos indicado,

este valor ya será el 1145. El resto de opciones las dejamos como aparecen por defecto y pulsamos “Finalizar”. Como queda configurada la conexión la podemos ver en la Figura 3.26.

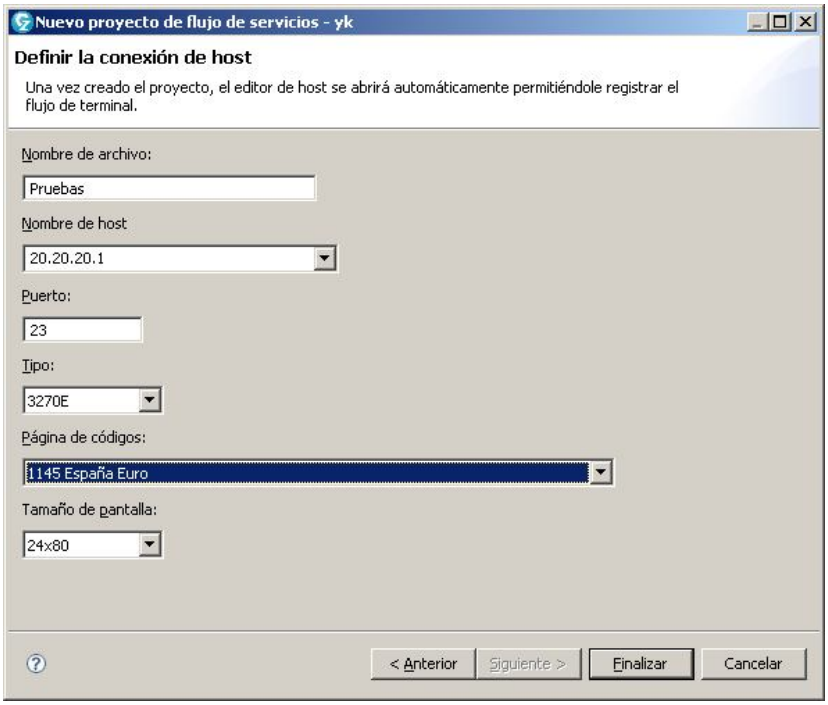


Figura 3.26 Conexión con el mainframe

Con esto ya hemos terminado de crear nuestro proyecto. Nos aparecerá la ventana de la Figura 3.27 donde marcamos las dos opciones para que no nos aparezcan más sugerencias y pulsamos “Correcto”.

Figura 3.27 Ventana de sugerencias

Automáticamente se nos abrirá en la parte superior derecha una pantalla con una conexión al mainframe. En este proyecto podemos grabar todos los flujos que queramos, no nos tenemos que limitar a un flujo por proyecto.

Para grabar un flujo, el RDz tiene que reconocer las pantallas con las que vamos a interactuar. Tenemos la opción de capturarlas por medio de RDz mientras grabamos el flujo, haberlas capturado previamente también por medio de RDz, o importar los mapas BMS que están en el

mainframe y que indican las propiedades que tiene una pantalla. BMS es el acrónimo de Basic Mapping Support y es una interfaz entre las aplicaciones CICS y las terminales, incluidas impresoras, que permite manejar las pantallas 3270 que se muestran.

En este caso, hemos decidido importar los mapas BMS ya que es el método que recomienda IBM. En otras ocasiones las capturaremos desde RDz. Para ello lo primero que hacemos es transferir por medio del BlueZone estos mapas, que se encuentran en el data set CICS.V3R2M0.CICS.SDFHSAMP, en modo ASCII a nuestra máquina. Para importarlos hacemos clic derecho sobre el nodo "*nombre_proyecto*.Terminal" en la parte de "Explorador de proyectos" y elegimos "Importar → BMS" como podemos ver en la Figura 3.28.

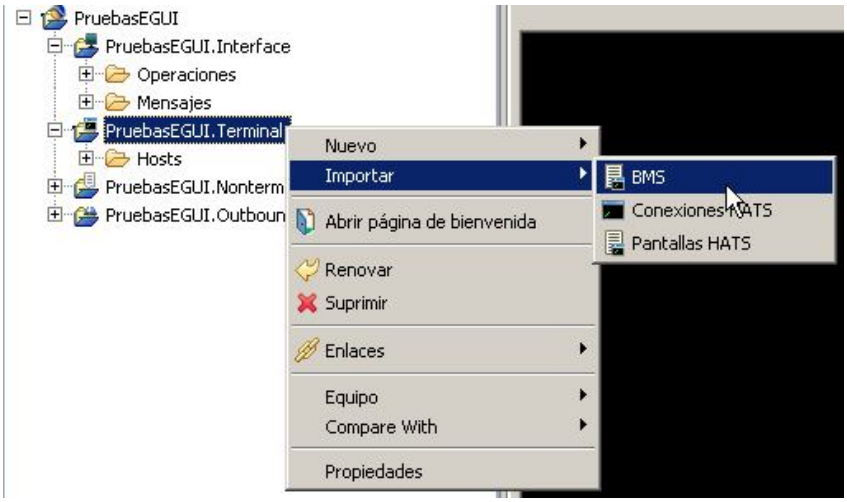


Figura 3.28 Importar mapas BMS

Aparecerá la pantalla de la Figura 3.29 donde pulsamos "Sistema de archivos…" y buscamos y elegimos los mapas que nos hemos descargado previamente.

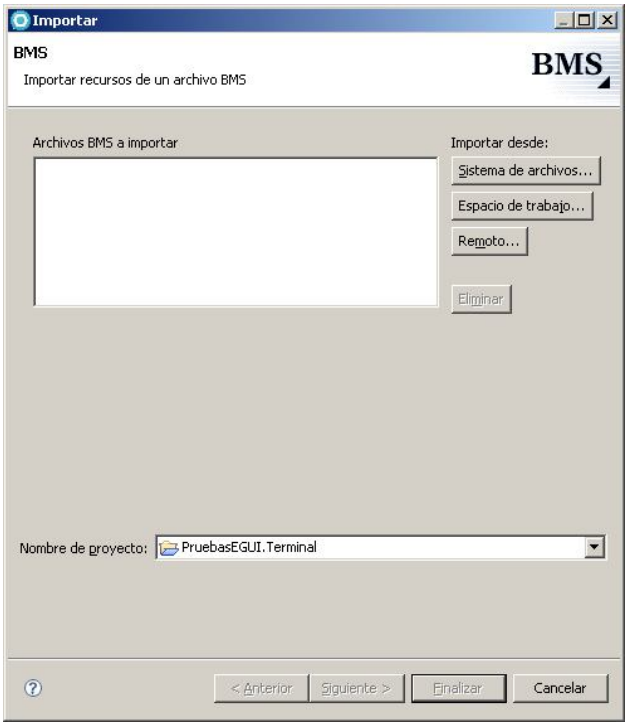


Figura 3.29 Selección de mapas a importar

Pulsamos “Siguiente” y nos aparecerá la ventana de la Figura 3.30 donde nos aseguramos que la opción “Página de códigos de host” sea la 1145 y pulsamos “Finalizar”.

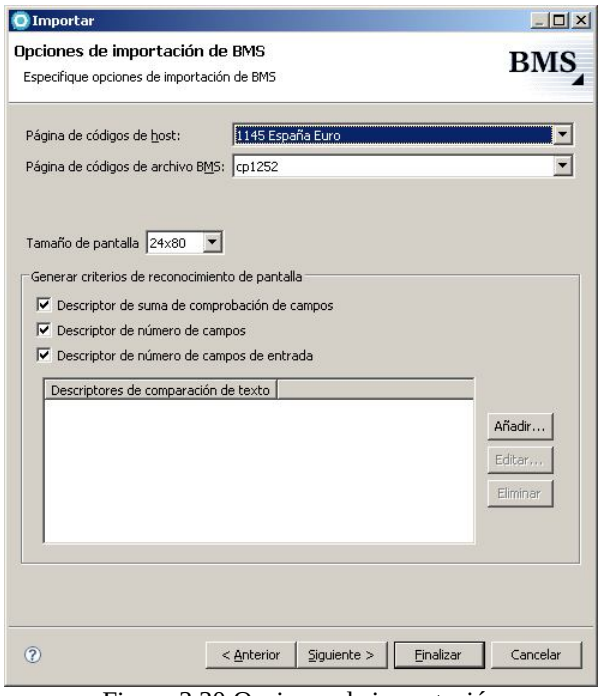


Figura 3.30 Opciones de importación

Con esto ya tenemos los mapas importados y todo preparado para poder empezar a grabar nuestro flujo.

3.3.2.1.- Captura de pantallas

En algunos casos puede que no tengamos acceso a los mapas de una transacción o que el RDz no reconozca correctamente un mapa importado con una pantalla. Cuando esto ocurre tenemos dos opciones para capturar la pantalla y que el RDz la reconozca, hacerlo previamente a la grabación del flujo o hacerlo durante la grabación del flujo.

Si queremos capturar las pantallas previamente, en la pantalla donde tenemos la conexión con el mainframe, en la parte derecha del RDz, vamos recorriendo todas las pantallas que vamos a recorrer a la hora de grabar nuestro flujo. En cada una de ellas pulsamos el botón de la Figura 3.31 para capturar la pantalla.


Figura 3.31 Botón de capturar pantalla

De esta forma aparecerá la ventana de la Figura 3.32 para que le demos un nombre a esta pantalla y pulsemos “Aceptar” para capturarla.

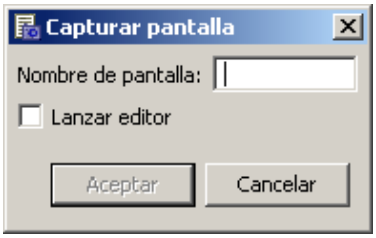


Figura 3.32 Dar nombre a la pantalla capturada

Si queremos capturar las pantallas a la vez que grabamos el flujo, simplemente empezamos a grabar el flujo como veremos más adelante sin hacer ningún paso previo. Cada vez que pasemos por una pantalla que RDz no reconozca por no tener capturada o por no haber importado su mapa nos aparecerá la ventana de la Figura 3.33.

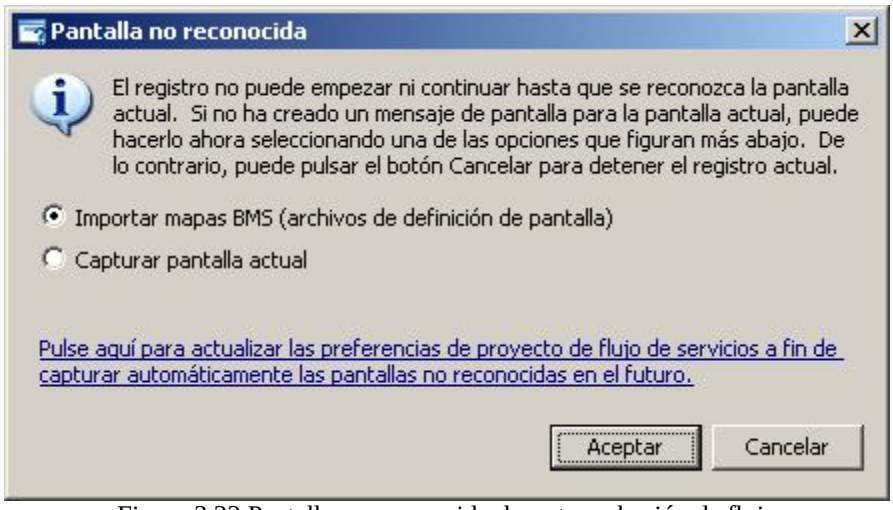


Figura 3.33 Pantalla no reconocida durante grabación de flujo

En este caso marcamos la opción "Capturar pantalla actual" y pulsamos "Aceptar". De esta forma nos aparecerá la ventana de la Figura 3.32 para que le demos un nombre a la pantalla y quedará capturada.

Dependiendo de las necesidades que tengamos podemos usar un método u otro. Como veremos más adelante, para grabar un bucle que involucre estas pantallas tendremos que modificarlas antes de empezar a grabar el flujo. Para ello tenemos que tener las pantallas capturadas previamente por lo que tendremos que utilizar el primer método o haber importado sus mapas BMS.

Una vez que hemos visto los diferentes métodos para que RDz reconozca las pantallas mientras grabamos un flujo, vamos a pasar a grabar el flujo en sí.

Para grabar un flujo nos dirigimos a la pantalla donde tenemos la conexión con el mainframe. El emulador que utiliza el RDz para conectarse al mainframe tiene las siguientes diferencias con respecto al Personal Communications que estábamos usando actualmente:

- Para borrar una pantalla usamos la tecla Esc en vez de la tecla Pausa
- Para simular el ENTER podemos usar la tecla Control o Intro del teclado numérico, como hasta ahora, o la tecla Intro habitual. IBM recomienda usar la tecla Control

Nos dirigimos a la pantalla que simula la terminal del mainframe y entramos en el CICS correspondiente que en este caso es el CICS1. Si aparece una pantalla de bienvenida o similar la borramos para tener una pantalla en blanco. Sobre esta pantalla, que podemos ver en la Figura 3.34, tecleamos la transacción que queremos lanzar para grabar el flujo que en este caso es la EGUI. Vemos como en la parte superior derecha nos dice "No se reconoce la pantalla" ya que no es una de las que hemos importado.

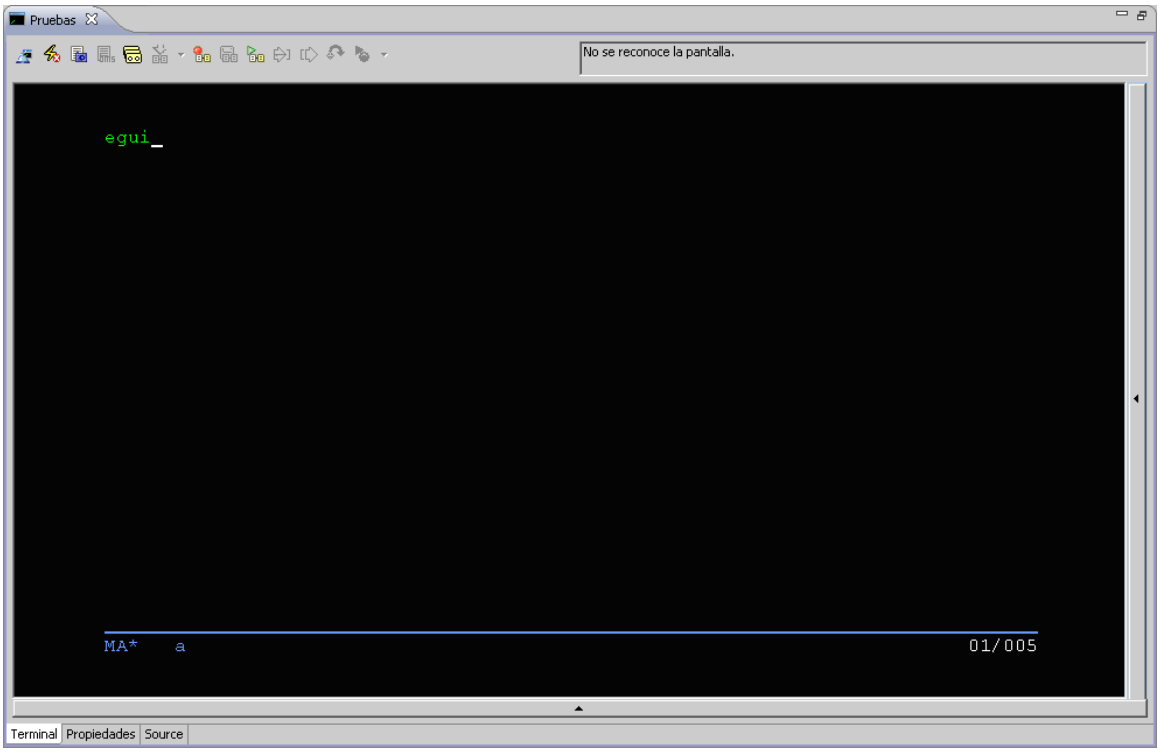


Figura 3.34 Pantalla en blanco donde tecleamos la transacción

Nos aparecerá la pantalla de la Figura 3.35 que es la primera pantalla de la EGUI. Vemos como en la parte superior derecha nos indica que la pantalla a sido reconocida. Este es el punto donde empezaremos a grabas nuestro flujo.

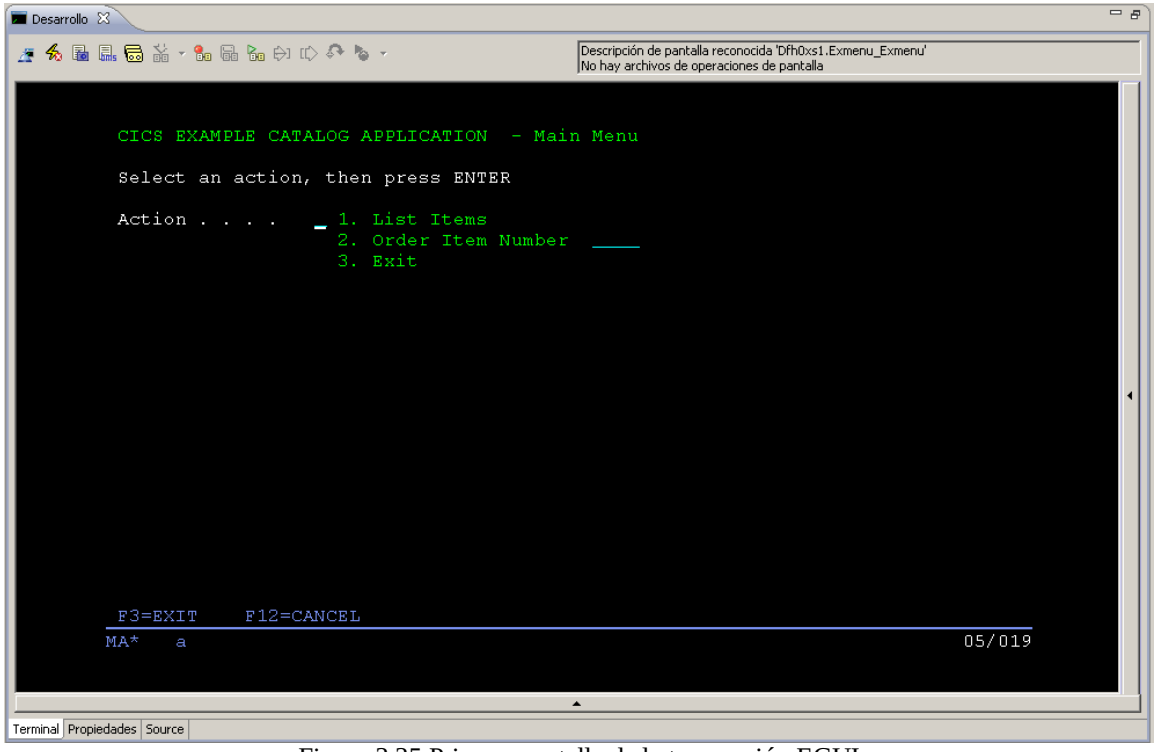


Figura 3.35 Primera pantalla de la transacción EGUI

Para ello pulsamos el botón de “Iniciar registro de flujo” que podemos ver en la Figura 3.36.

Figura 3.36 Botón de "Iniciar registro de flujo"

Aparece la pantalla de la Figura 3.37 donde le damos el nombre que queremos al flujo que vamos a grabar y donde seleccionamos la opción "Crear todos los recursos nuevos" para crear nuevos grupos de variables para este flujo y pulsamos "Finalizar".

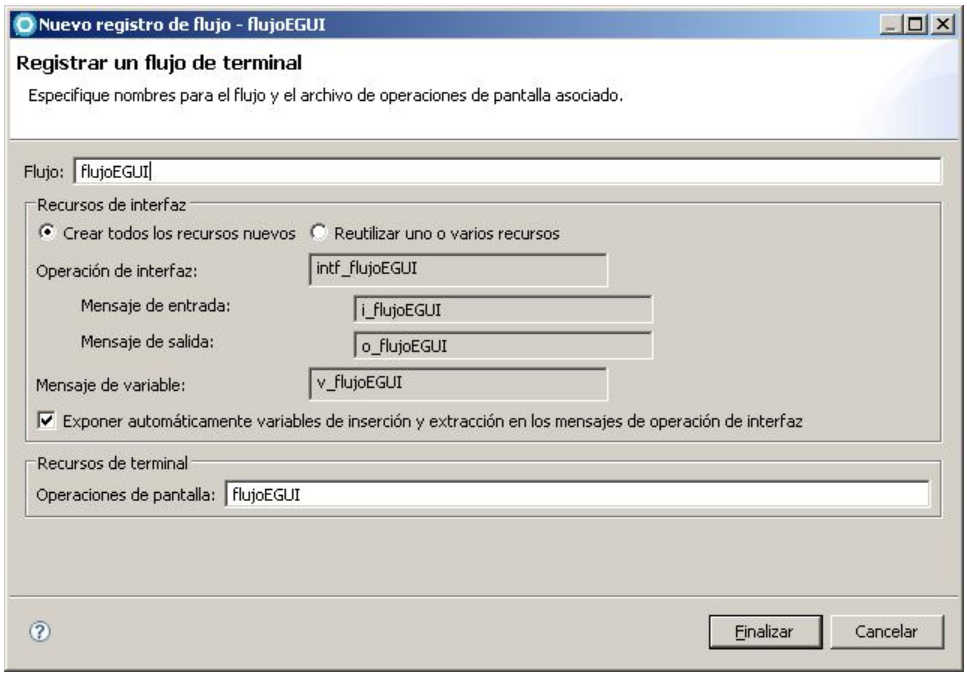


Figura 3.37 Nuevo registro de flujo

En la parte superior derecha nos aparecerá el mensaje de la Figura 3.38 que nos indica que el flujo está siendo grabado. También indica que la pantalla se ha reconocido.

Figura 3.38 Mensaje de flujo grabándose

En esta pantalla vamos a crear dos variables de entrada. La primera será para introducir la acción que queremos realizar mientras que la segunda indicará el elemento del que queremos consultar los detalles. Realmente nos valdría con crear sólo la variable para el artículo ya que la acción queremos que sea siempre la 2. Creamos dos variables para que luego podamos probar como llamar al flujo con varias variables de entrada y para poder simular algunos errores como introducir la acción 1.

Para crear una variable de entrada pulsamos el botón que podemos ver en la Figura 3.39.

Figura 3.39 Botón "Insertar datos en la pantalla"

Aparecerá la ventana de la Figura 3.40 para que podamos crear una nueva variable o seleccionar una creada anteriormente.

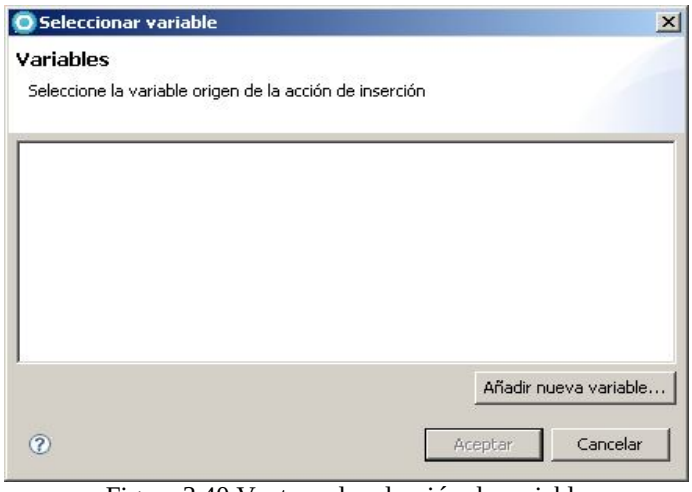


Figura 3.40 Ventana de selección de variable

En caso de tener alguna variable creada nos aparecería aquí pero como no las tenemos vamos a crear una nueva. Para ello, pulsamos el botón "Añadir nueva variable…" y nos aparece la pantalla de la Figura 3.41 para poder elegir el nombre de la variable.

Figura 3.41 Ventana para añadir nueva variable

Le damos el nombre de "accion" y pulsamos "Aceptar". Evitamos las tildes en los nombres de las variables por seguridad.

Esto nos lleva a la ventana anterior donde ya se ha añadido nuestra variable y vemos en la Figura 3.42 que está seleccionada. Pulsamos "Aceptar".

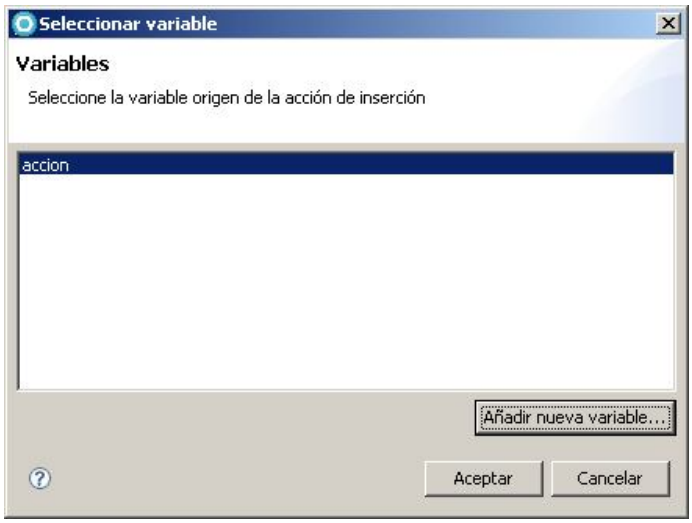


Figura 3.42 Variable creada ya seleccionada

Nos llevará a la pantalla donde estábamos. Si desplazamos el cursor sobre ella nos irá subrayando en rojo todos los campos que encuentra como campos de entrada. Haremos clic derecho sobre el campo de entrada al que queremos ligar la variable que en este caso será el campo de "Action" como se puede ver en la Figura 3.43. Cuando hayamos ligado la variable este campo de entrada quedará permanentemente subrayado en rojo.

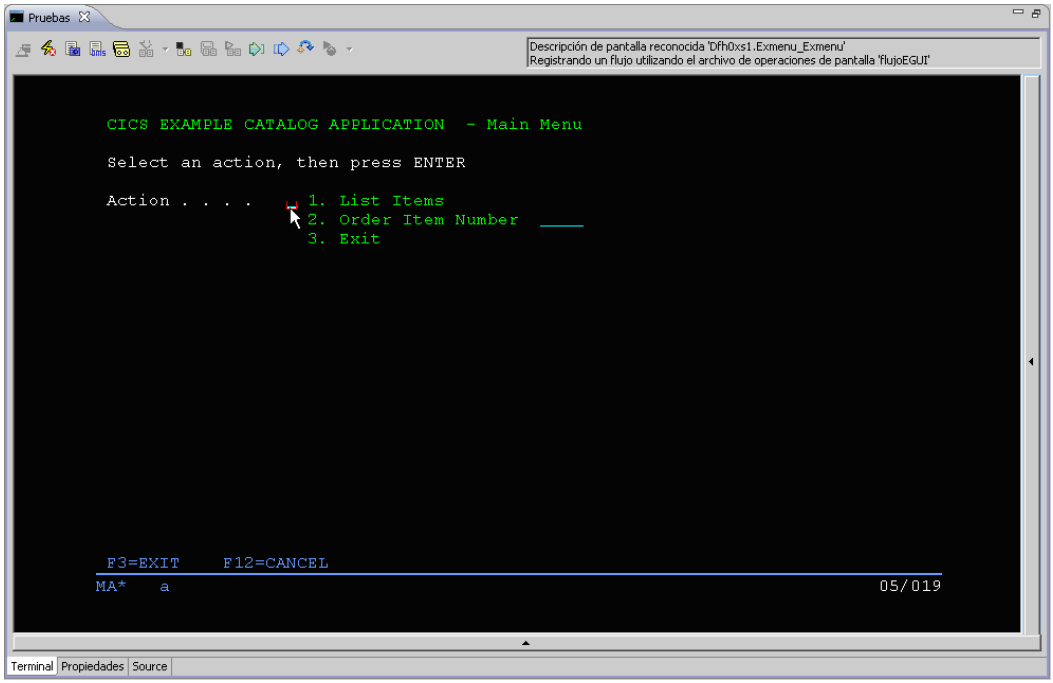


Figura 3.43 Campo "Action" ligado a variable accion

Repetiremos el mismo proceso para definir la variable de entrada "articulo" para poder introducir el número del artículo que queremos mostrar por pantalla. En la Figura 3.44 vemos como ambos campos quedan subrayados en rojo al estar ligados a una variable de entrada.

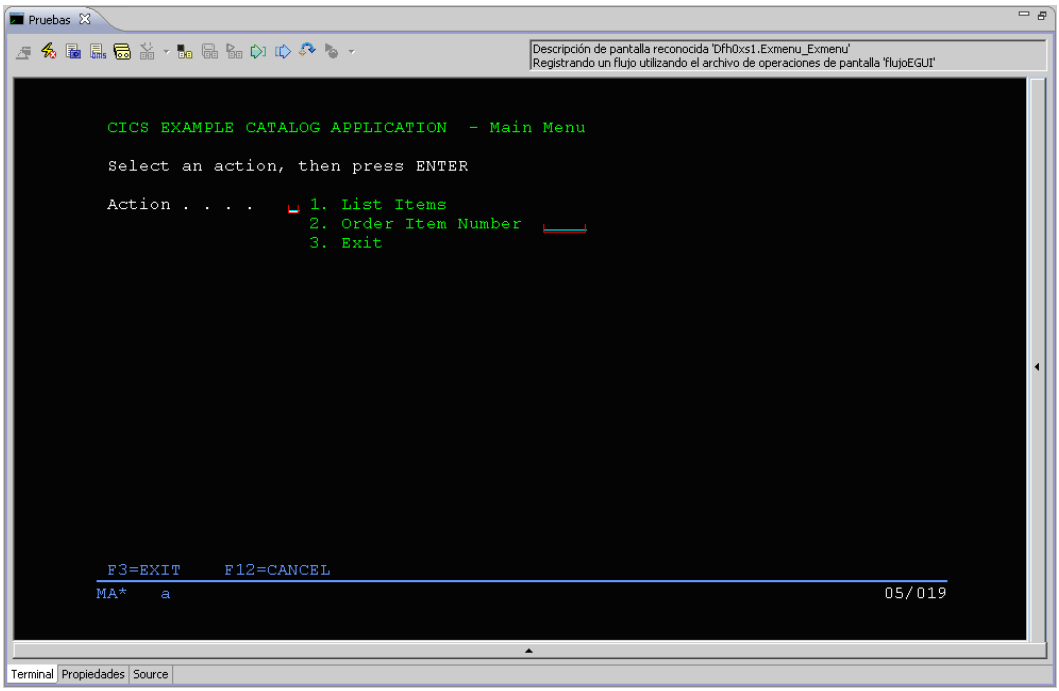


Figura 3.44 Campos "Action" y "Order Item Number" ligados a variables de entrada

Una vez que tengamos definidas todas las variables de entrada introducimos en acción un 2 y en el número de artículo uno cualquiera, por ejemplo el 0010, y pulsamos Control de forma que nos lleve a la pantalla de la Figura 3.45. Cuando se invoque el flujo lo que hará el SFR será introducir el contenido de la variable "accion" en el campo "Action", el de la variable "articulo" en "Order Item Number" y enviar una señal de ENTER que es lo que simula la tecla Control.

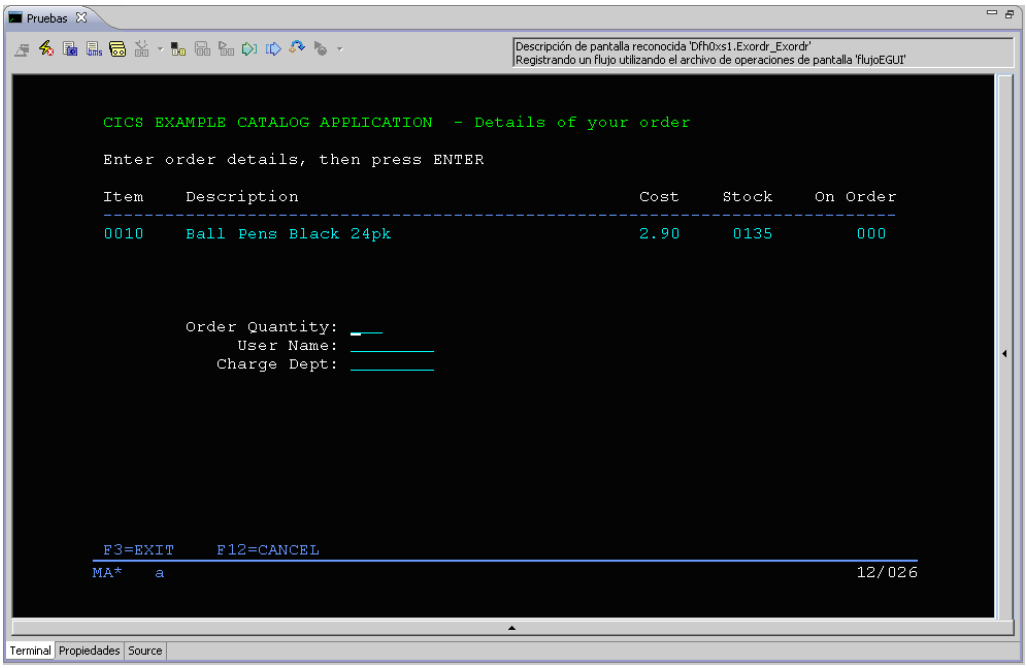


Figura 3.45 Siguiente pantalla de la transacción EGUI

En esta pantalla vamos a definir una variable de salida que extraiga la información que se nos muestra del item introducido. Para definir una variable de salida primero recuadramos con el ratón el área que queremos que sea de salida de forma que quede como podemos ver en la Figura 3.46.

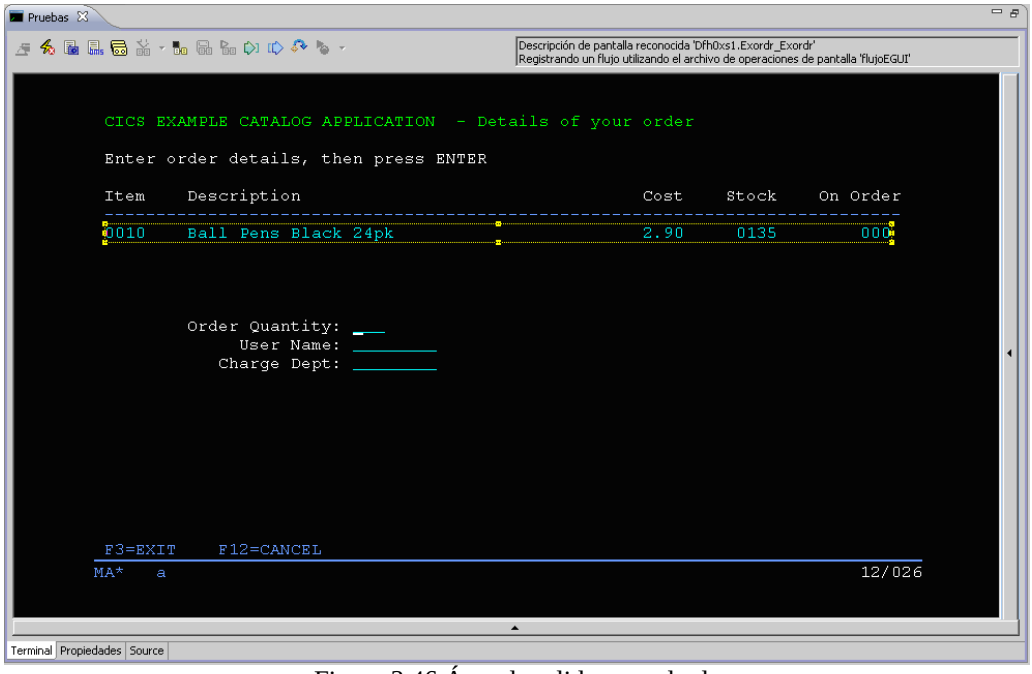


Figura 3.46 Área de salida recuadrada

Para definir una variable pulsamos el botón de la Figura 3.47.

Figura 3.47 Botón "Extraer datos de la pantalla"

Nos aparece la ventana de la Figura 3.48 donde podemos seleccionar una variable ya creada o crear una nueva. También podemos configurar esta variable. En especial podemos decir las dimensiones del área que queremos extraer y si la queremos extraer en una variable simple de tipo matriz en caso de tener más de una fila, en general escogeremos la primera opción. Una opción interesante es la de definir delimitadores de fila y campo. Esto hace que, si por ejemplo definimos el ";" como delimitador de fila y el "|" como delimitador de campo, cuando nos envíe estos datos de salida insertará estos símbolos después de cada fila o campo. Esto nos facilita saber donde acaba una fila o campo para poder trocear el stream de datos de salida que recibamos.

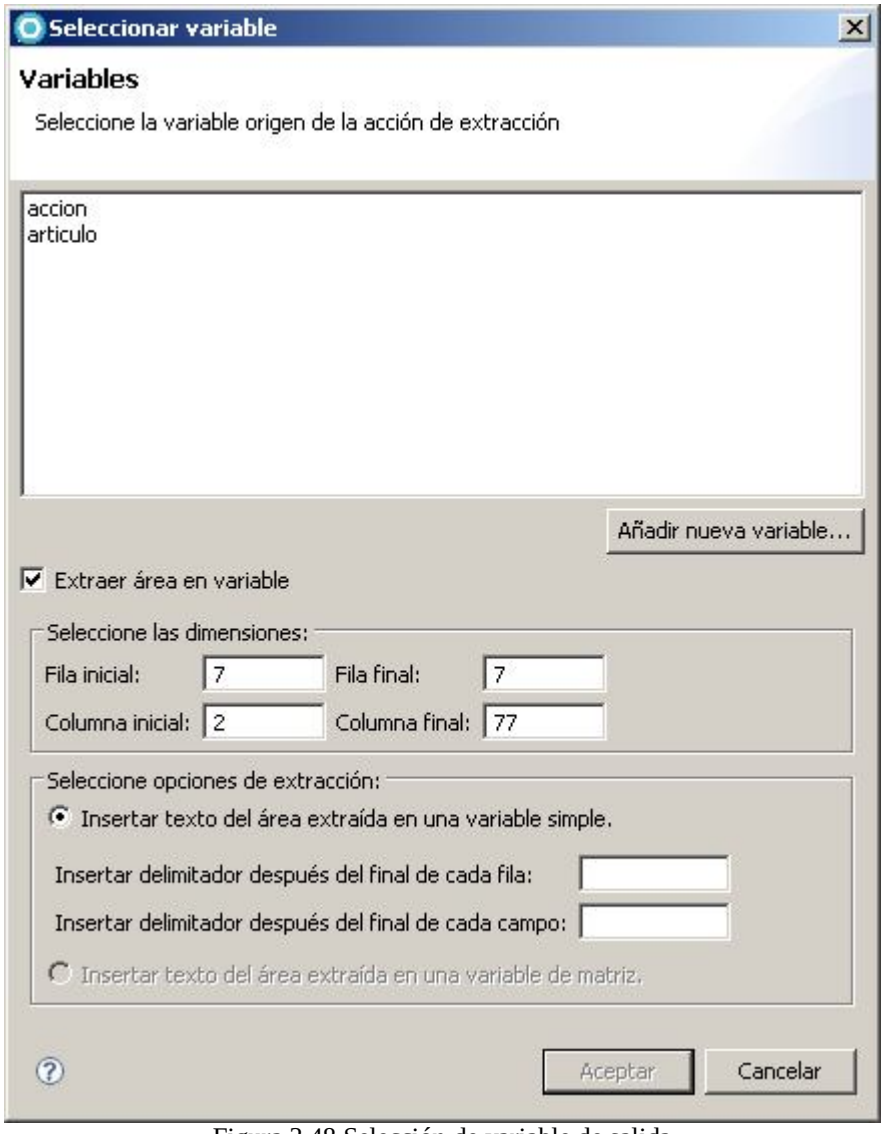


Figura 3.48 Selección de variable de salida

Como queremos crear una nueva variable pulsamos el botón "Añadir nueva variable…". Nos aparecerá la ventana de la Figura 3.41 para que podamos darle el nombre a la variable. La

llamamos “salida” y pulsamos “Aceptar”. Nos aparece la ventana anterior donde nos ha añadido la nueva variable ya seleccionada. Pulsamos “Aceptar” para que nos lleve a la pantalla de la Figura 3.49 donde vemos que el área que teníamos remarcada como de salida nos ha cambiado a rojo.

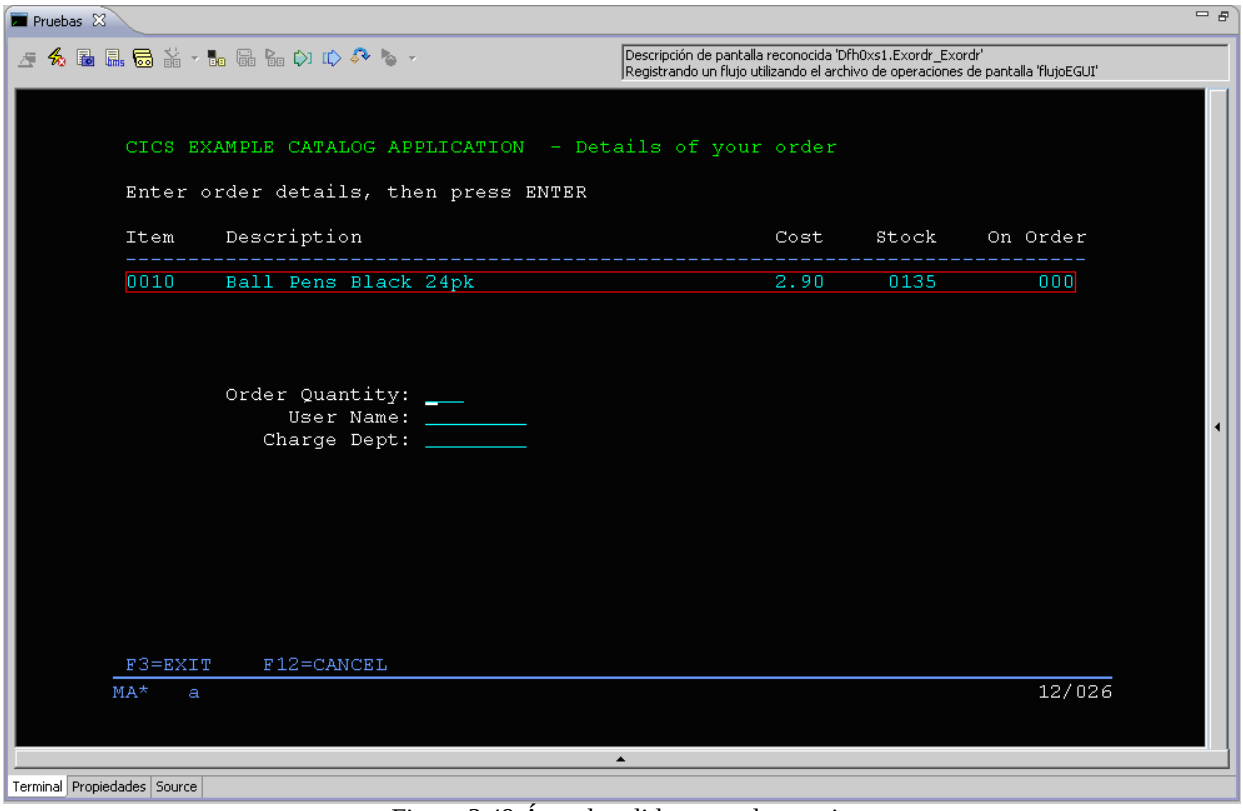


Figura 3.49 Área de salida marcada en rojo

Aquí vamos a terminar la grabación de nuestro flujo por lo que pulsamos el botón de la Figura 3.50 para finalizar la grabación.


Figura 3.50 Botón para finalizar la grabación de un flujo

Se nos habilita el botón de la Figura 3.51 que pulsaremos para guardar el flujo.


Figura 3.51 Botón para guardar el flujo

Ahora que tenemos grabado y guardado nuestro flujo vamos a comprobar que realiza las acciones que nosotros queremos y que devuelve los datos de salida adecuados. Para ello vamos a hacer una simulación del flujo.

Lo primero que hacemos es cargar el flujo que queremos reproducir. Para ello pulsamos sobre la flecha del botón de la Figura 3.52.


Figura 3.52 Botón para cargar flujos

Nos aparecerán los flujos que tenemos grabados, que en este caso sólo será uno, y elegimos el que queramos reproducir como vemos en la Figura 3.53.

Figura 3.53 Selección del flujo que queremos cargar

Cargar un flujo de esta manera también nos servirá como veremos más adelante para crear flujo con alternativas.

Una vez que tengamos el flujo cargado, si volvemos a pinchar sobre la flecha del botón de la Figura 3.52 veremos que nos vuelve a aparecer el listado de los flujo con el flujo cargado marcado con un ✓.

Para empezar a simular un flujo nos situamos con el emulador de conexión en la primera pantalla del flujo que en este caso es la pantalla inicial de la transacción EGUI que podemos ver en la Figura 3.35. Pulsamos el botón de la Figura 3.54 para empezar a simular el flujo.


Figura 3.54 Botón para iniciar simulación de un flujo

Aparecerá una barra de progreso que indica que se esta cargando el flujo. A continuación aparecerá la ventana de avisos que podemos ver en la Figura 3.55.

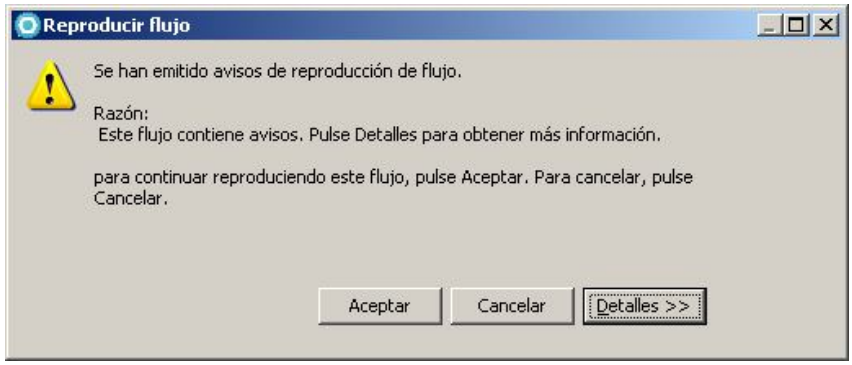


Figura 3.55 Ventana de avisos al simular un flujo

Los avisos que nos muestra no son importantes por lo que pulsamos “Aceptar”. Esto hará que aparezca la ventana de la Figura 3.56 donde podemos darle valores a las variables de entrada que hemos definido.

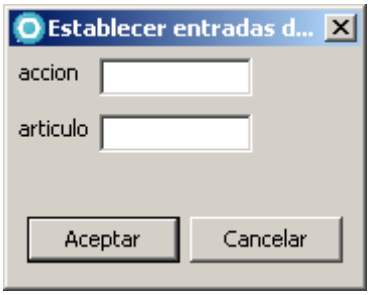


Figura 3.56 Ventana para introducir valores de las variables de entrada

A la variable “accion” le damos el valor 2 ya que es la única acción para la que hemos grabado un flujo. A la variable “articulo” le damos un valor distinto al 0010 pare ver que funciona

correctamente para cualquier item que introduzcamos. Concretamente introducimos el valor 0020.

Empezará a simularse el flujo. Si estamos atentos podemos ver como va pasando por las pantallas que hemos grabado introduciendo los valores de las variables. Al final se nos mostrará una ventana similar a la de la Figura 3.57 donde aparecerán las variables de salida que hayamos definido junto al valor que han tomado. En este caso vemos como nos devuelve los datos del item 0020. Estos datos son los mismos que obtenemos si realizamos manualmente en el CICS los pasos que hemos grabado en el flujo.

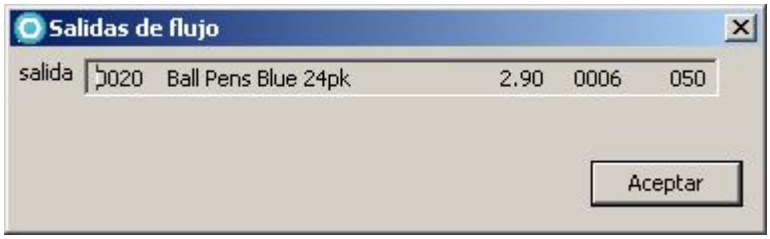


Figura 3.57 Ventana con variables de salida

Si pulsamos “Aceptar” podemos ver la pantalla a la que hubiésemos llegado si hubiésemos hecho estos pasos de forma manual que es similar a la de la Figura 3.40 pero con los datos del item 0020.

Si intentamos reproducir un flujo sin haber cargado uno previamente, o si no estamos en la pantalla donde hemos empezado a grabar el flujo, nos aparecerá un mensaje como el que podemos ver en la Figura 3.58.

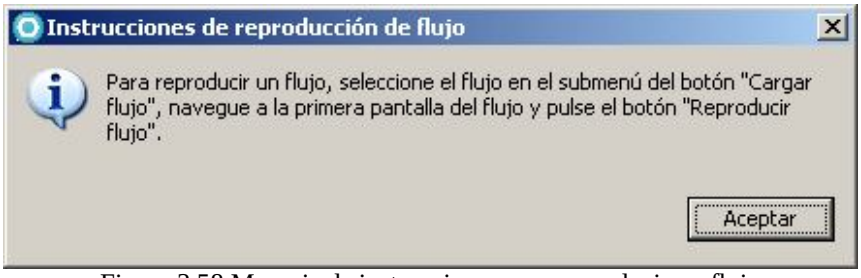


Figura 3.58 Mensaje de instrucciones para reproducir un flujo

Si en vez de introducir el valor 2 en la variable “accion” introducimos el 1, el flujo nos llevará a una pantalla correcta pero que no hemos grabado en nuestro flujo. Esto hará que se muestre un error similar al de la Figura 3.59 ya que se excede un timeout que introduce RDz por seguridad intentando reconocer una pantalla que no tiene grabada en el flujo.

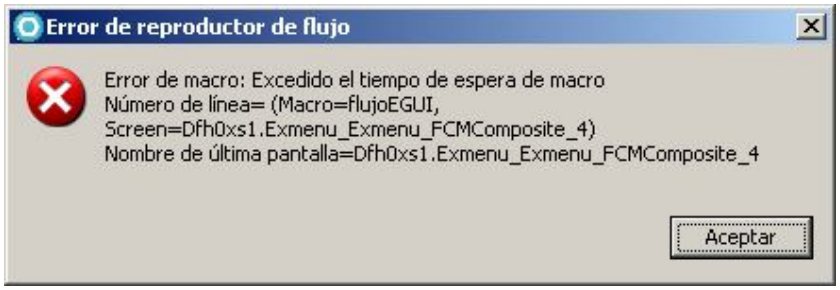


Figura 3.59 Error de pantalla no reconocida

Con esto ya tenemos grabado y probado nuestro primer flujo. El siguiente paso es generar los archivos necesarios e instalarlo en el mainframe para poder hacerle peticiones.

3.3.3.- Instalación del flujo

El primer paso para generar los archivos necesarios que tenemos que instalar en el mainframe es generar un archivo de propiedades para el flujo. Para ello hacemos clic derecho sobre el flujo que hemos creado, que es el archivo con extensión seqflow en la carpeta "Flujos", y elegimos "Nuevo → Archivo de propiedades de generación" como vemos en la Figura 3.60.

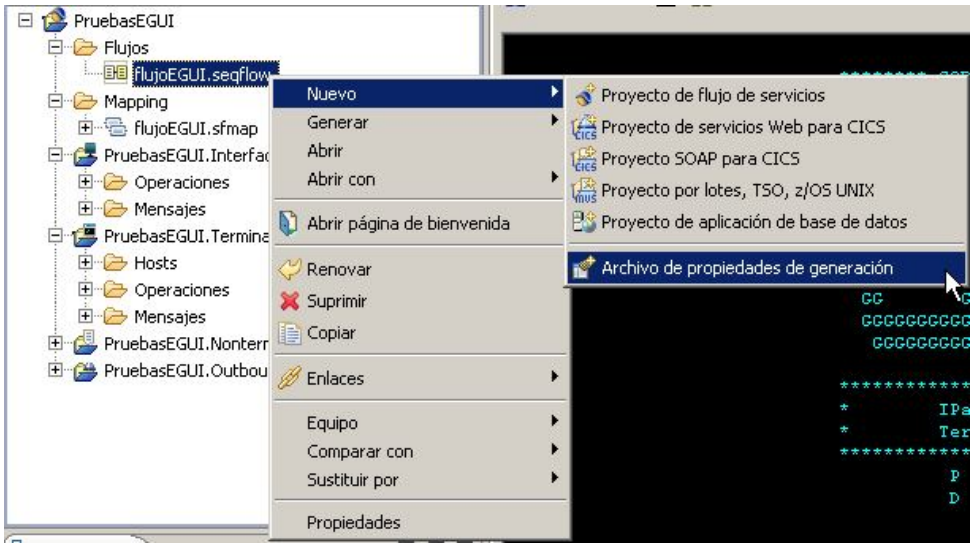


Figura 3.60 Generar archivo de propiedades de generación para un flujo

Nos aparece la ventana de la Figura 3.61 donde podemos elegir el nombre del archivo. Lo dejamos con el nombre que sale por defecto y pulsamos "Siguiente".

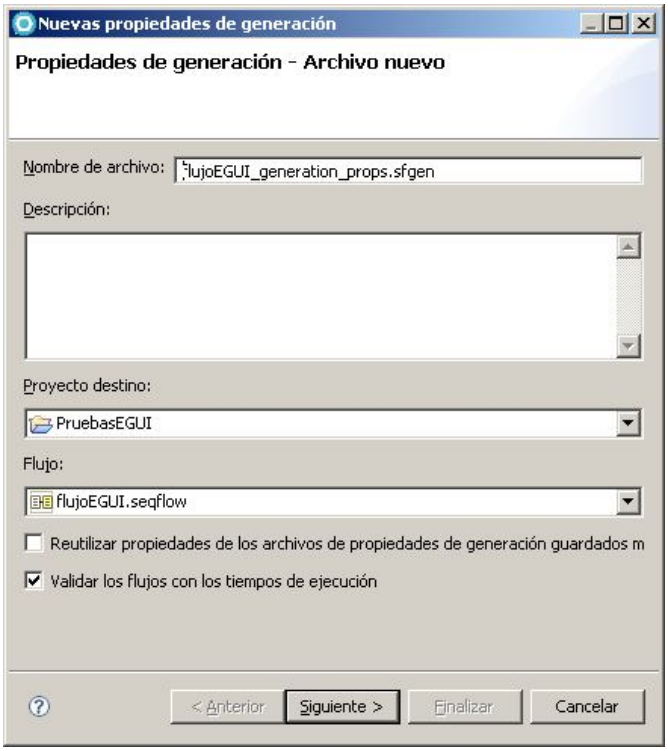


Figura 3.61 Nombre del archivo de propiedades de generación

En la siguiente pantalla elegimos la versión de CICS TS para la que queremos crear el flujo que en nuestro caso es la 3.2. Dependiendo de si tenemos la versión 7.5 o 7.6 del RDz nos aparecerá la ventana de la Figura 3.62 o 3.63 respectivamente.

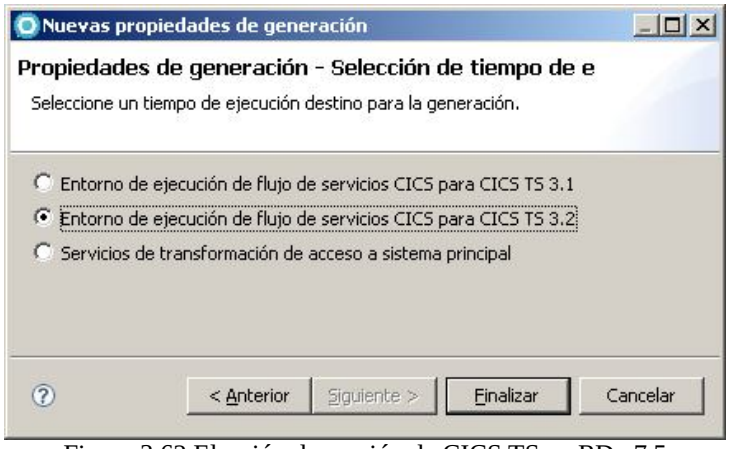


Figura 3.62 Elección de versión de CICS TS en RDz 7.5

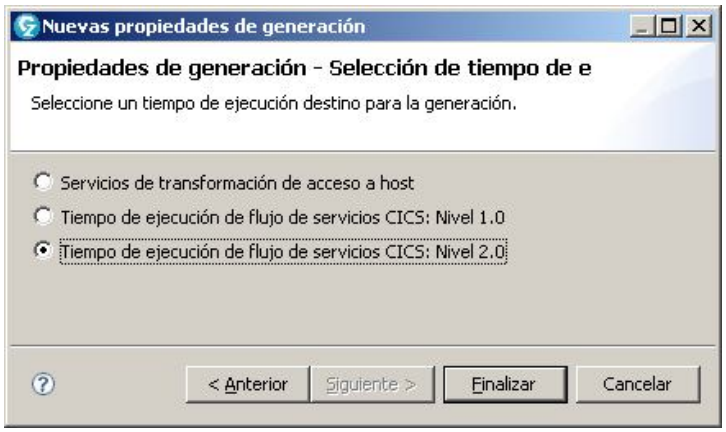


Figura 3.63 Elección de versión de CICS TS en RDz 7.6

En la Figura 3.62 está claro la opción que tenemos que elegir. En la Figura 3.63 elegimos el Nivel 2.0 ya que es el adecuado para nuestro SFF y CICS según podemos ver en la Tabla 3.1.

Nivel de runtime:	**CICS Service Flow Feature (Service Flow Runtime):**	**CICS Transaction Server para z/OS:**
Tiempo de ejecución de flujo de servicios CICS: Nivel 1.0	CICS Service Flow Runtime para CICS Transaction Server para z/OS V3.1	CICS Transaction Server para z/OS V3.1 **sin** APAR PK83534
Tiempo de ejecución de flujo de servicios CICS: Nivel 2.0	CICS Service Flow Runtime para CICS Transaction Server para z/OS V3.2	• CICS Transaction Server para z/OS V3.1 **con** APAR PK83534 • CICS Transaction Server para z/OS V3.2 • CICS Transaction Server para z/OS V4.1

Tabla 3.1 Nivel de runtime adecuado según el SFF y CICS

Esta pantalla es la única diferencia que vamos a encontrar entre las versiones 7.5 y 7.6 del RDz en las tareas que nosotros vamos a realizar.

Una vez que hayamos elegido el Nivel o versión de CICS TS adecuado pulsamos "Finalizar". Nos lleva directamente a una pantalla similar a la de la Figura 3.64 donde vamos a indicar los valores que hay que cambiar en las propiedades de generación. Algunos parámetros no aparecerán al principio pero lo harán conforme vayamos configurando el archivo.

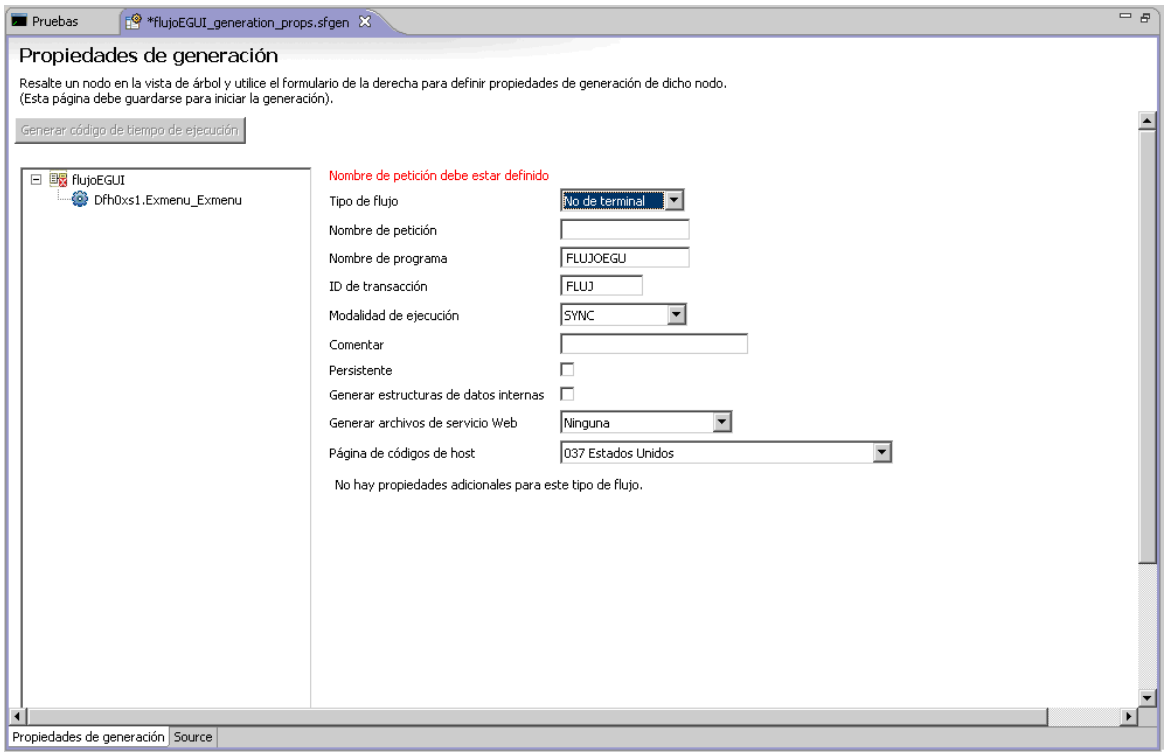


Figura 3.64 Archivo de propiedades de generación

Los parámetros que vamos a modificar y los valores que les vamos a dar son los siguientes:

- Tipo de flujo: Link3270 Bridge
- Nombre de petición: EGUI1PR
- Nombre de programa: EGUI1PG
- ID de transacción: EG1T
- Generar estructuras de datos internas: Seleccionado
- Generar archivos de servicio Web: Servicios Web para CICS
- Página de códigos del host: 1145 España Euro
- Tecla PF inicial: CLEAR
- Datos de transacción de arranque: EGUI
- URI de punto final: http://20.20.20.1:13311/pruebaegui
- Nombre de archivo WSBIND: pruebaegui
- Nombre de archivo WSDL: pruebaegui
- Vía de acceso a archivo HFS WSDL: /repositorio/wsdir/EGUI/prueba

Guardamos el proyecto con el botón de la Figura 3.65 que está en la barra de menús.


Figura 3.65 Botón para guardar los cambios.

Una vez que hayamos guardado los cambios se nos habilitará el botón de la Figura 3.66 en la parte superior del archivo de propiedades de generación y lo pulsamos. De esta forma generaremos todos los elementos que necesitamos para instalar el flujo en el mainframe.

Figura 3.66 Botón para generar los elementos necesarios para instalar el flujo en el mainframe

Una vez que pulsemos este botón nos aparecerá la ventana de la Figura 3.67. En ella marcaremos la opción "Generar en otro ubicación" para poder subir directamente los archivos necesarios al mainframe. Esto lo podríamos hacer por medio del BlueZone pero de esta forma es más sencillo y rápido. En la "Información de control de trabajo" le indicamos nuestro identificador de usuario para que nos despliegue los componentes necesarios en nuestros data sets. También configura los JCLs que ejecutaremos para que nos avise a nosotros cuando finalice la ejecución. También marcamos las dos opciones para que cree dos JCLs para compilar el programa y definir en el CICS los recursos necesarios.

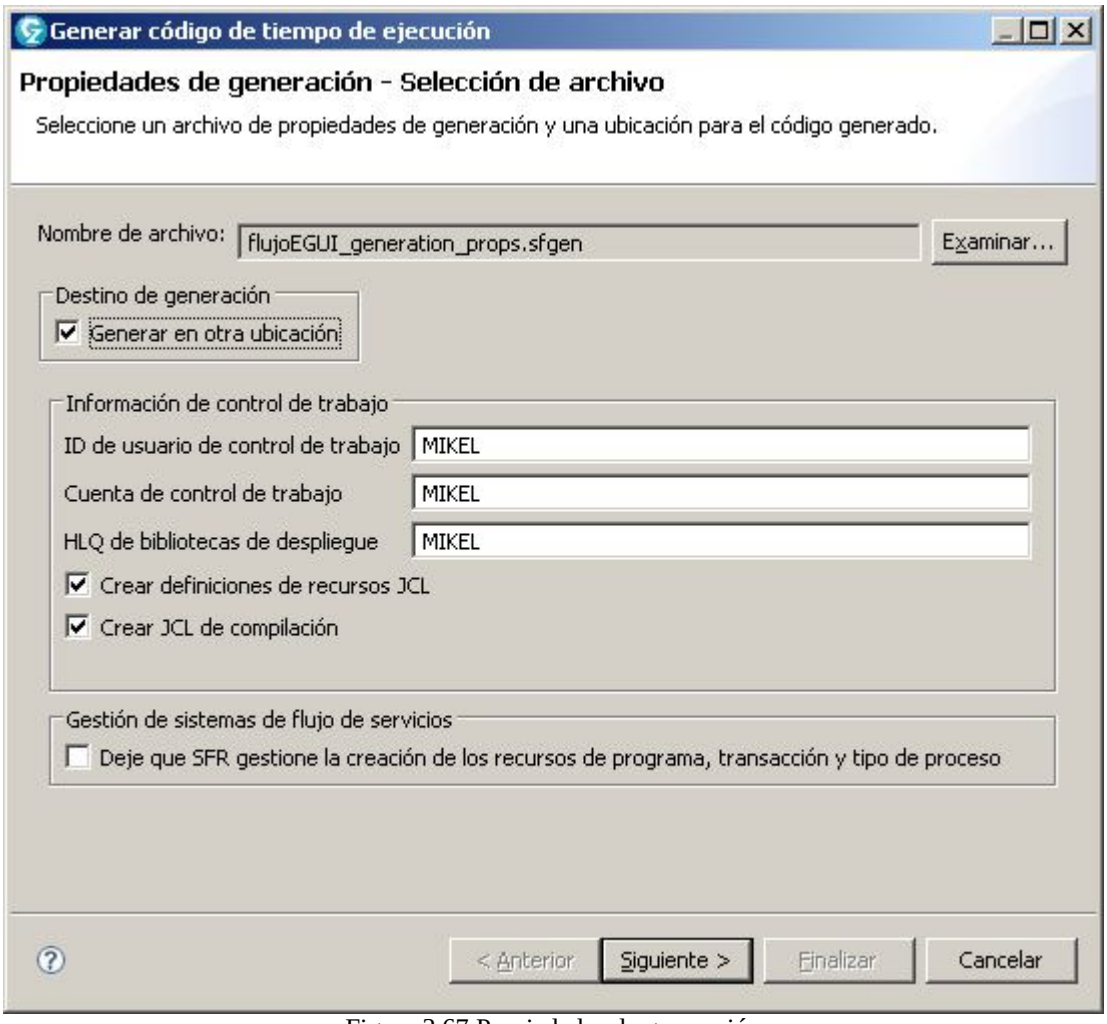


Figura 3.67 Propiedades de generación

Pulsamos "Siguiente". Si no hemos importado las plantillas JCL o estas han cambiado desde que las hemos importado, nos aparecerá la pantalla de la Figura 3.68 avisándonos de este hecho. Si las queremos importar hacemos clic en el enlace que nos aparece y las importamos como hemos explicado anteriormente.

Figura 3.68 Mensaje que aconseja importar las plantillas JCL

Tras pulsar "Aceptar" en la pantalla anterior, si nos ha aparecido, nos aparece la pantalla de la Figura 3.69 que nos deja elegir en que lugar generar los archivos. Elegimos "generar en una ubicación remota" y si solo hemos creado una conexión nos la elegirá automáticamente. Si tuviésemos más de una, por ejemplo a más de un CICS, podríamos elegir que conexión usar para subir los archivos.

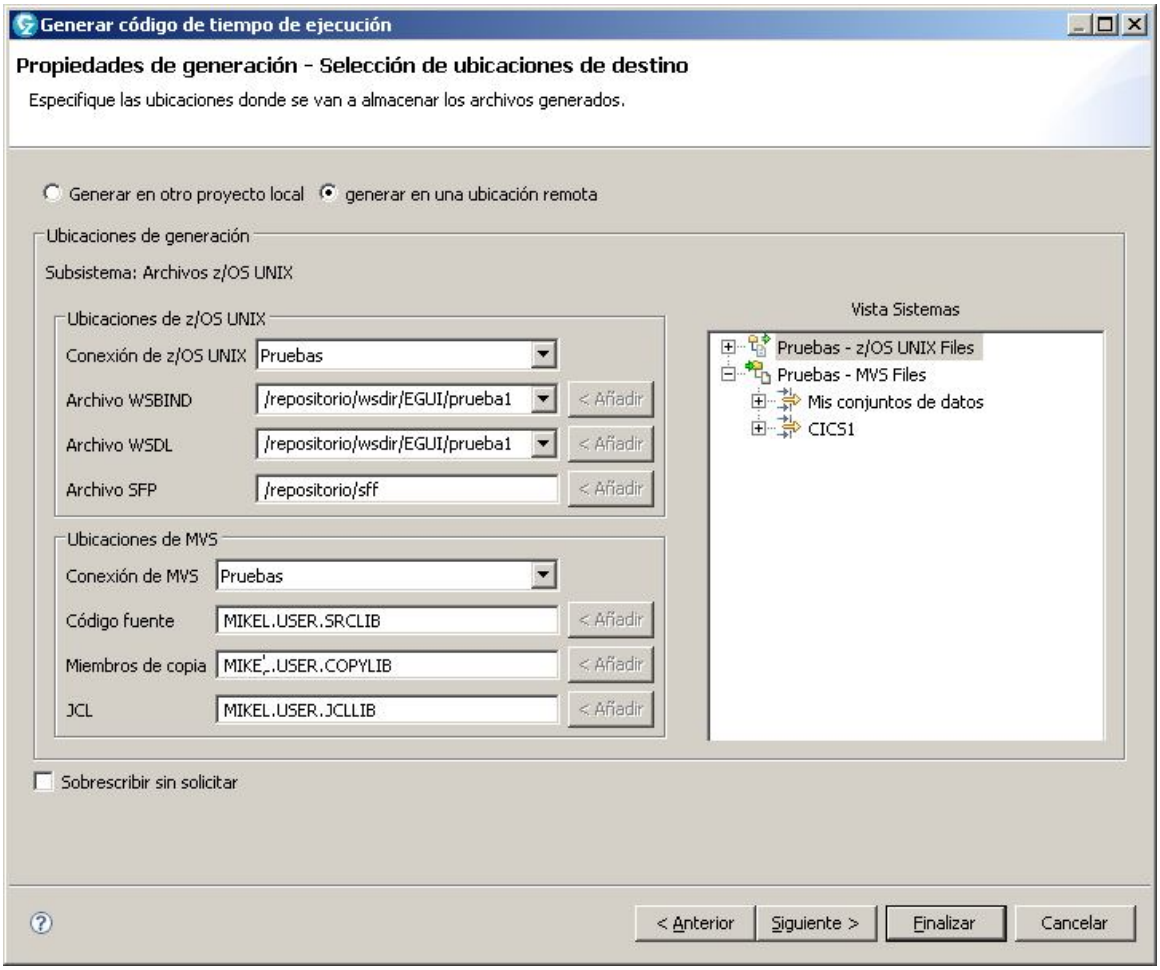


Figura 3.69 Ubicación de destino de los archivos a generar

Si alguno de los archivos aparece sin una ruta donde va a ser subido, navegamos por el árbol de directorios hasta llegar a la ubicación donde lo queremos subir y tras seleccionar esta ubicación pulsamos el botón "Añadir" que se encuentra al lado de cada archivo.

La mayoría de los archivos los podemos desplegar donde queramos excepto el "Archivo SFP". Este lo tenemos que subir al directorio "/repositorio/sff" ya que es el que hemos indicado en la configuración del SFF donde la transacción CMAN va a buscar los flujos que se pueden instalar.

Cuando tengamos todos los archivos en su ubicación podríamos pulsar "Siguiente" para que nos aparezcan más opciones de generación pero como ya no necesitamos modificar nada más pulsamos "Finalizar"

En la parte inferior de la pantalla de la Figura 3.69 aparecerá una barra de progreso donde se nos irá informando de los pasos que se están realizando como podemos ver en la Figura 3.70.

Figura 3.70 Barra de progreso de generación de archivos

Cuando este proceso acabe nos aparecerá una ventana indicándonos los resultados que se han producido al realizar las diferentes tareas de generación y despliegue de los archivos. Si todo se ha realizado correctamente nos aparecerá una pantalla similar a la Figura 3.71.

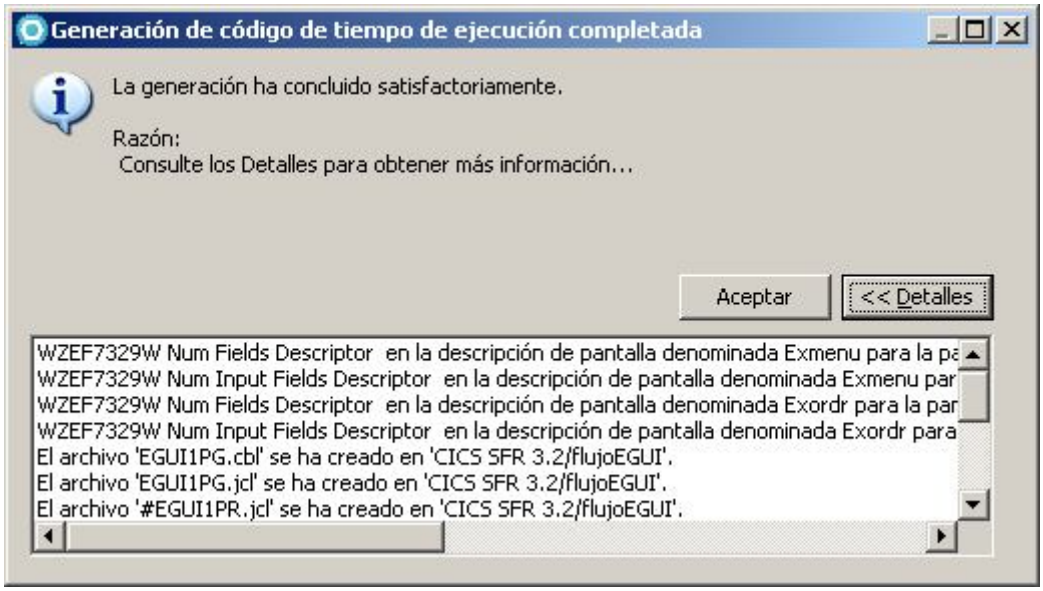


Figura 3.71 Tareas de generación finalizadas correctamente

Con esto ya tendríamos en el mainframe todos los archivos necesarios para poder instalar el flujo. Vemos que en el árbol de archivos del proyecto han aparecido los archivos que se han generado y posteriormente subido al mainframe. Estos archivos generados los podemos ver en la Figura 3.72.

El archivo cbl es el programa COBOL que se ejecutará cuando invoquemos al flujo.

Los archivos jcl permiten definir los recursos necesarios en el CICS y compilar el programa COBOL respectivamente.

El archivo sfp es el archivo de propiedades del flujo que leerá la transacción CMAN para instalar el flujo.

El archivo sfgen es el archivo de propiedades de generación que hemos configurado anteriormente.

Los archivos wsdl y wsbin son necesarios para poder invocar el flujo desde un servicio Web.

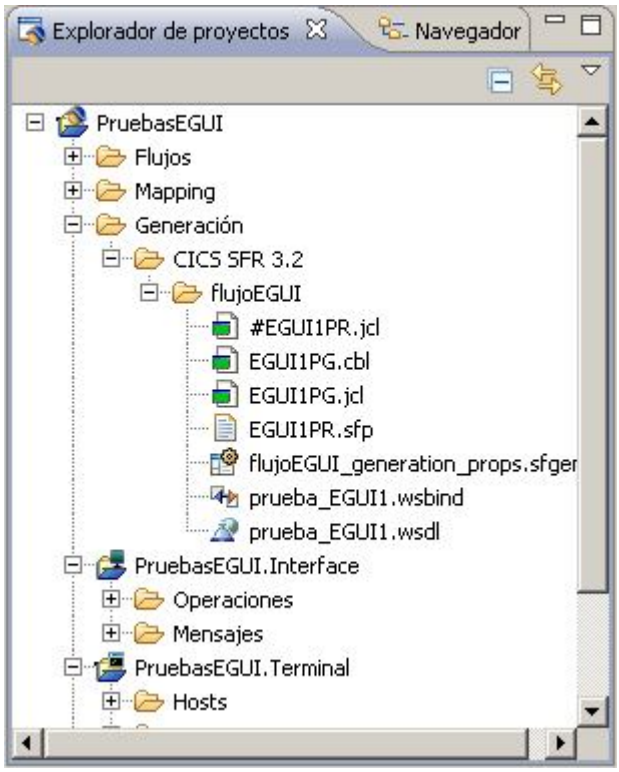


Figura 3.72 Archivos generados

Nos dirigimos ahora al mainframe para crear una PIPELINE, realizar unas pequeñas modificaciones en los archivos JCL transferidos al mainframe y submitirlos.

Primero abrimos una conexión con el CICS para poder definir la PIPELINE y el grupo donde se van a definir todos los recursos que se van a utilizar. Definir una PIPELINE sólo es necesario si queremos crear un servicio Web para invocar al flujo.

Una vez que estemos conectados y autenticados por medio de la transacción CESN ejecutamos el comando CEDA DEF PIPELINE(EGUIPIP1) G(SFFEGUI).

El nombre de la PIPELINE y del GRUPO podría ser cualquier otro.

Nos aparece la pantalla de la Figura 3.73 donde tenemos que modificar algunos campos.

```
DEF PIPE(EGUIPIP1) G(SFFEGUI)
OVERTYPE TO MODIFY                                      CICS RELEASE = 0650
 CEDA  DEFine PIpeline( EGUIPIP1 )
  PIpeline      ==> EGUIPIP1
  Group         ==> SFFEGUI
  Description   ==>
  STatus        ==> Enabled           Enabled | Disabled
  Respwait      ==> Deft              Default | 0-9999
  Configfile    ==>
  (Mixed Case)  ==>
                ==>
                ==>
                ==>
  SHelf         ==> /var/cicsts/
  (Mixed Case)  ==>
                ==>
                ==>
                ==>
  Wsdir           :
  (Mixed Case)    :
```

Figura 3.73 Pantalla de definición del recurso PIPELINE

Estos campos y sus valores son:

- Configfile: /usr/lpp/cicsts/cicsts32/samples/pipelines/basicsoap11provider.xml
- SHelf: /var/cicsts/provider/
- Wsdir: /repositorio/wsdir/EGUI/prueba/

Pulsamos Control y vemos como en la parte inferior de la pantalla nos aparece un mensaje indicándonos que la definición se ha realizado correctamente como nos pasaba cuando definíamos los recursos para las conexiones del CTG.

Los valores de Configfile y SHelf son siempre los mismos independientemente del flujo que creemos. El valor de Wsdir cambia para apuntar al directorio donde hayamos subido los archivos WSBIN y WSDL de nuestro servicio Web.

Ahora nos dirigimos a una sesión de TSO para modificar los JCL y submitirlos.

Para ello primero abrimos una sesión de TSO y nos dirigimos al data set donde hayamos subido los JCL's. Si tenemos muchos miembros, ejecutamos el comando "SORT CHA" de forma que los miembros se nos ordenen por orden descendente de fecha de modificación. De esta forma los últimos miembros modificados, que probablemente sean los JCLs que hemos subido, aparezcan en la parte superior de la lista.

Abrimos el miembro ÑEGUI1PR en edición. Este JCL se utiliza para definir los recursos de PROCESSTYPE, TRANSACTION y PROGRAM que necesita el flujo para funcionar. Será al PROCESSTYPE al que tengamos que invocar para que se ejecute el flujo. Modificamos en la cabecera del JCL las partes que están en amarillo en la Figura 3.74 hasta dejarlo como aparece en la Figura mencionada. Esto hará que el JCL aparezca en la cola de salida con el nombre ÑEGUI1PR y que se avise cuando finalice su ejecución al usuario de TSO que lo ha mandado ejecutar.

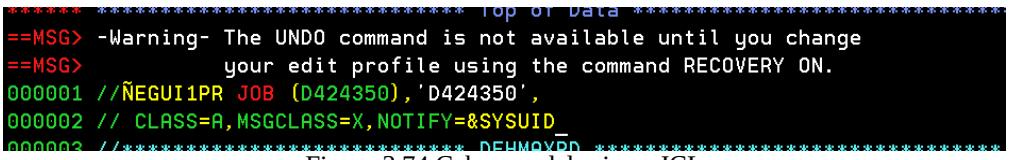


Figura 3.74 Cabecera del primer JCL

También tenemos que modificar el grupo donde se definen los recursos ya que por defecto lo hacen en el grupo CICSSFRG. Para que se definan en el grupo SFFEGUI donde hemos definido la PIPELINE ejecutamos el comando "c CICSSFRG SFFEGUI all". De esta forma cambiamos todas las apariciones de CICSSFRG por SFFEGUI.

Submitimos el JCL ejecutando desde la línea de comandos el comando "SUB".

En la parte inferior nos aparecerá el mensaje de la Figura 3.75 indicándonos que el JCL ha sido submitido.

Figura 3.75 Mensaje de JCL submitido

Cuando acabe su ejecución nos mostrará un mensaje similar al de la Figura 3.76 donde MAXCC=0 indica que se ha ejecutado sin fallos.

Figura 3.76 Ejecución de JCL correcta

Vamos a modificar y submitir ahora el segundo JCL que sirve para compilar el programa COBOL que se ejecuta al invocar el flujo.

En este JCL sólo modificamos la cabecera hasta dejarla similar al caso anterior por las mismas razones, como podemos ver en la Figura 3.77.

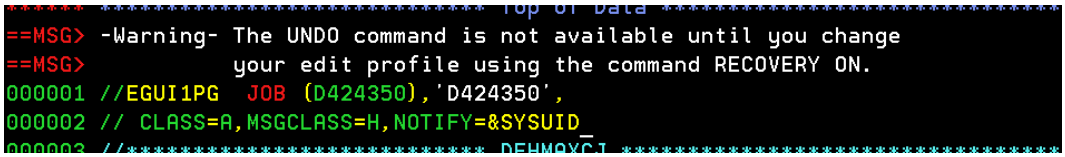


Figura 3.77 Cabecera del segundo JCL

Los submitimos y vemos su resultado. En la Figura 3.78 vemos como ahora tenemos un MAXCC=4.

Figura 3.78 Ejecución de JCL con avisos

Un MAXCC=4 indica que se han producido mensajes de advertencia e información al ejecutar el JCL. Por experiencia sabemos que en este caso el 4 es aceptable ya que son mensajes de información que nos dicen que hay opciones en algunas estructuras IF que no se pueden dar en ningún caso por lo que se van a eliminar.

Una vez ejecutados ambos JCLs nos dirigimos al CICS e instalamos todo el grupo para que nos instale los recursos necesarios para poder ejecutar el flujo. Esto lo hacemos ejecutando el comando “CEDA INSTALL G(SFFEGUI)”.

En la parte inferior aparecerá un mensaje como el de la Figura 3.79 diciendo que se ha instalado correctamente el grupo.

Figura 3.79 Instalación correcta de grupo

Ejecutamos la transacción CMAN que sirve para gestionar los flujos. Esta transacción permite gestionar los flujos cuyo archivo sfp se encuentre en la carpeta /repositorio/sff que es la que se definió en la instalación del SFF para tal efecto. Un ejemplo de pantalla de esta transacción la podemos ver en la Figura 3.80.

Una vez en esta pantalla pulsamos F2 para que instale los nuevos flujos que enentre en la carpeta /repositorios/sff. A continuación pulsamos Control para que actualice la lista de flujos y veremos como el EGUI1PR aparece entre ellos.

```
DFHMA00      CICS TS V3.2 Service Flow Runtime - Installed Service Flows

Request name filter: *                                              More:
Actions: 1= View details 2= Enable 3= Disable 4= Delete

 Act  Request name    Status      Use count   Response
  _      CESNPR       Unusable    0
  _      EGUIPT1      Enabled     0
  _      EGUI1PR      Enabled     0
  _      PRUEBAP      Unusable    0
  _      PRUEBAPR     Enabled     0
  _      PRUFRAOP     Unusable    0
  _      PRUGCAOP     Unusable    0
  _      PRUR         Enabled     0
  _      SFFEXAM0     Enabled     0
  _      SFFEXAM1     Enabled     0
  _      TFNPROC      Enabled     0
  _      TFN1PROC     Enabled     0
```

Figura 3.80 Pantalla de la transacción CMAN

Vemos como nuestro flujo, el EGUI1PR, se encuentra con estado Enabled lo que significa que puede ser usado.

Para probarlo, nos dirigimos de nuevo a RDz y hacemos clic derecho sobre el archivo WSDL que generamos anteriormente. Nos dirigimos a "Servicios Web → Probar con explorador de servicios Web" como podemos ve en la Figura 3.81.

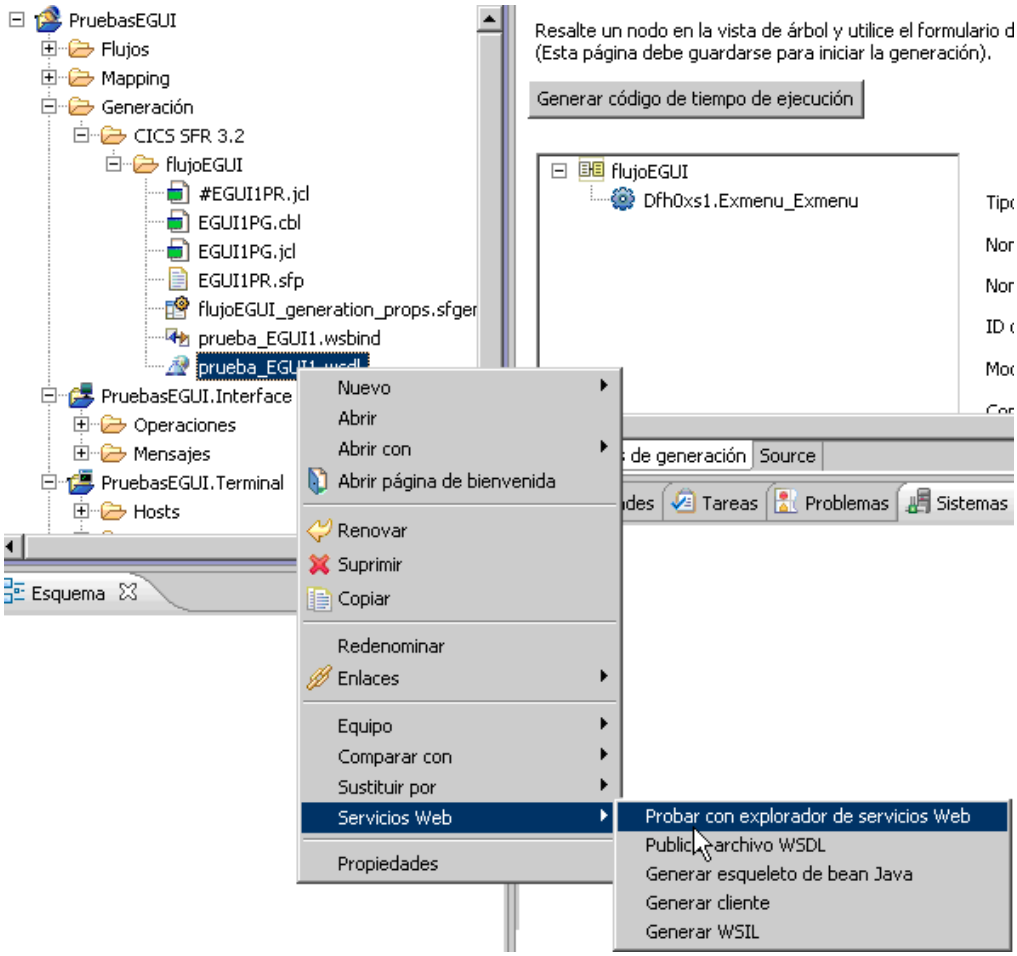


Figura 3.81 Probar flujo desde servicios Web

Nos aparece una ventana similar a la de la Figura 3.82. De esta forma vamos a poder probar el flujo desde un servicio Web. Para ello tendremos que tener definido el recurso PIPELINE que hemos visto anteriormente y el recurso TCPIPSERVICE que veremos en el apartado 3.5 como definirlo. Desplegamos por completo las opciones de la pestaña "Navegador" y hacemos clic sobre el nodo que pone "DFHMADPLOperation". De esta forma aparece en la pestaña derecha unos campos que tenemos que rellenar para realizar la petición al servicio Web.

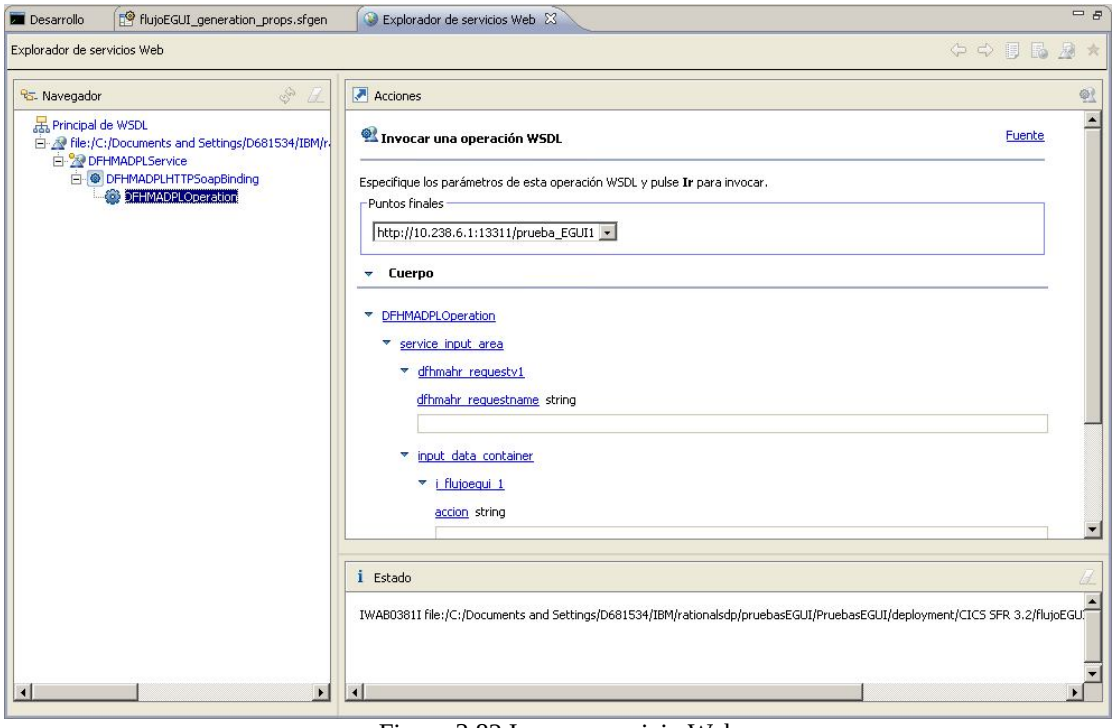


Figura 3.82 Invocar servicio Web

En especial rellenamos con los siguientes valores que podemos ver en la Figura 3.83.

- dfmahr_requestname: EGUI1PR
- accion: 1
- articulo: 0030, podría ser cualquier otro

Figura 3.83 Datos del flujo a invocar

El valor de dfmahr_requestname es el que hayamos definido en "Nombre de la petición", la acción es en este caso la 2 y el artículo cualquiera de los que están definidos en el maiframe, que en este caso va del 0010 al 0210. Si tuviésemos mas variables de entrada aparecerían aquí. Pulsamos en "Ir" para realizar la petición al servicio Web.

Vemos que tras unos segundo, en la parte inferior, bajo la pestaña "Estado" habremos recibido un mensaje que es una representación del sobre SOAP de respuesta con los datos que ha devuelto el servicio Web. En la Figura 3.84 podemos ver la salida que se produce.

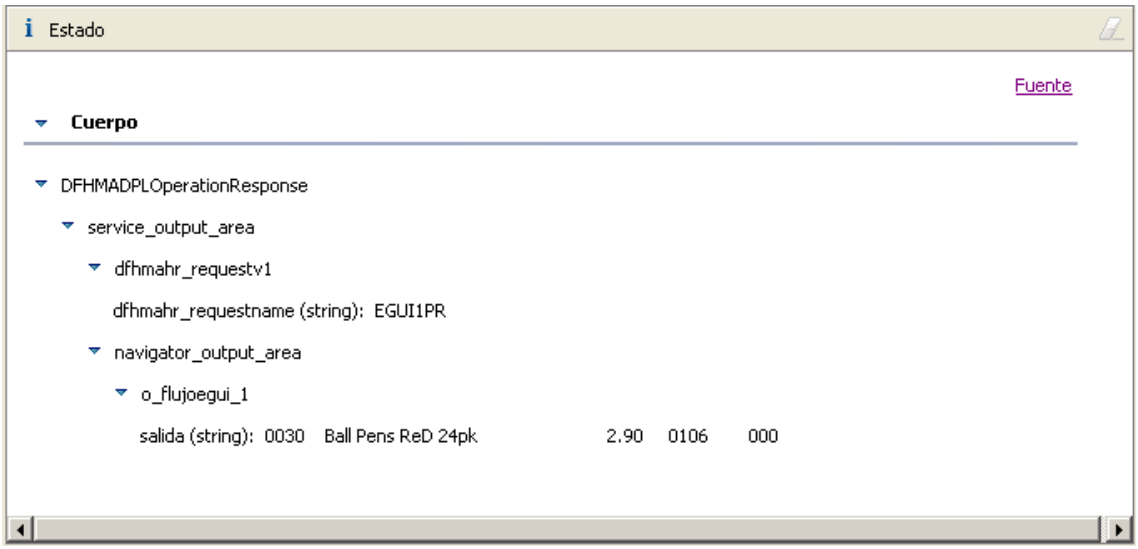


Figura 3.84 Mensaje SOAP de respuesta

En la última línea podemos ver los datos devueltos por el flujo. Estos datos son los mismos que si ejecutamos desde el CICS la acción 2 de la transacción EGUI para el artículo 0030.

Si tuviésemos más de una línea de salida podríamos haber elegido a la hora de crear la variable de extracción hacerlo en una variable de tipo matriz. De esta forma, nos aparecería una línea por cada fila de salida que tuviésemos.

Si hubiésemos elegido delimitadores de fila y campo los símbolos ";" y "|" respectivamente obtendríamos algo similar a lo que podemos ver en la Figura 3.85.

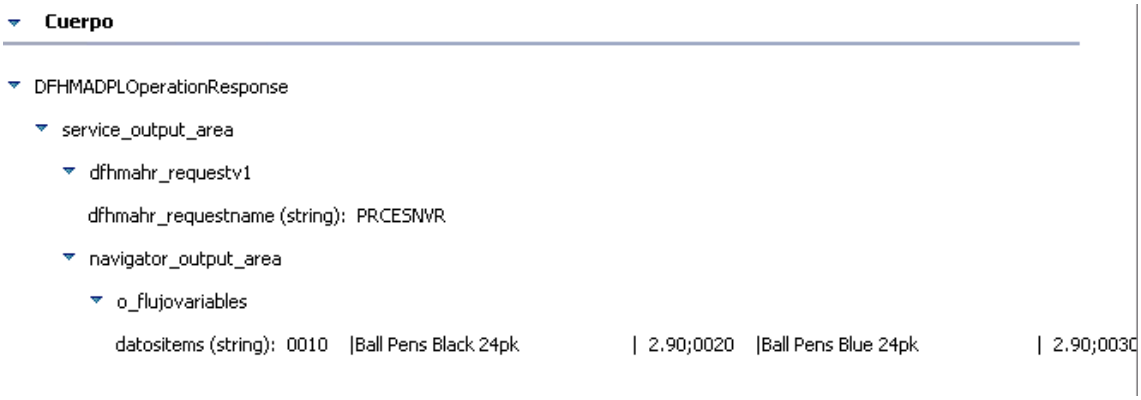


Figura 3.85 Salida con delimitadores de fila y campo

Si nos equivocamos pidiendo un artículo que no existe o marcando la acción 1 en el estado nos aparecerá el mensaje que podemos ver en la Figura 3.86 que nos indica que se ha producido un error ejecutando el flujo, normalmente porque hemos llegado a una pantalla que no hemos grabado.

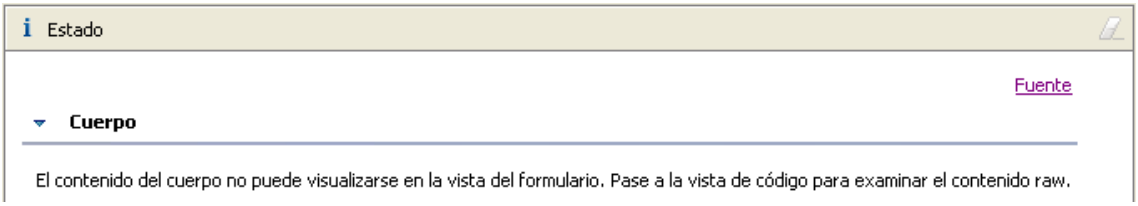


Figura 3.86 Mensaje SOAP de error en la petición

Si hacemos clic sobre "Fuente" para poder ver los mensajes SOAP de envío y respuesta, en el de respuesta podemos ver el mensaje que vemos en el Documento 3.2.

```
<SOAP-ENV:Envelope xmlns:SOAP-
        ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:q0="http://www.xmlenabli.com/schemas/xmlenablIInterface"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <SOAP-ENV:Body>
                <SOAP-ENV:Fault>
                <faultcode>SOAP-ENV:Server</faultcode>
                <faultstring>Conversion to SOAP failed</faultstring>
                <detail>
                        <CICSFault
                                xmlns="http://www.ibm.com/software/htp/cics/WSFau
                                lt">DFHPI1008 10/08/2010 09:42:13 CICS1 00709
                                SOAP message generation failed because of incorrect
                                input (INPUT_STRUCTURE_TOO_SMALL ).</CICSFault>
                </detail>
                </SOAP-ENV:Fault>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Documento 3.2 Mensaje SOAP de error al invocar un flujo

Aunque ponga "INPUT_STRUCTURE_TOO_SMALL" que parece que nos indica que faltan variables de entrada, por experiencia sabemos que este mensaje se produce cuando el flujo llega a una pantalla que no ha grabado.

Si al hacer una petición a un flujo, ya sea de esta forma o por medio del CTG o de servicios Web como veremos a continuación, podemos capturar en el CICS el flujo en modo debug para ver los pasos que se van ejecutando y ver donde falla.

Para hacer esto nos conectamos al CICS. Ejecutamos el comando "CEDF *ID_transacción"*. El nombre de la transacción será aquel que hayamos indicado en el campo "ID de transacción" del archivo de propiedades de generación del flujo correspondiente. De esta forma, cuando se arranque la transacción, que será cuando hagamos una petición del flujo, esta se capturará en modo debug. Utilizando las teclas que se nos indican por pantalla podemos ver los pasos que se van ejecutando como los mapas de las pantallas que se ejecutan, la información que se introduce en estas pantallas, etc. De esta forma podemos ver donde falla nuestro flujo y arreglarlo, ya sea editándolo o creando uno nuevo.

Una vez que hayamos corregido el flujo, salimos del modo debug ejecutando el comando "CEDF *ID_transacción*,OFF".

3.3.4.- Comandos problemáticos en los flujos

En el archivo de propiedades de generación, hemos visto como en el tipo de flujo hemos puesto "Link3270 Bridge". Si nos fijamos en la página 20 del manual SC34-6830-03 - CICS External Interfaces Guide v3.2, nos encontramos con una serie de consideraciones a la hora de programar para Link3270. En especial, en la página 22 nos encontramos con lo siguiente:

Security Processing
When a bridge facility is created, it is signed on as a preset USERID terminal, with the client's USERID. As with other preset terminals, the SIGNON and SIGNOFF commands are not permitted, and INVREQ is raised.

Esto quiere decir que cuando se usa el Link3270 Bridge se crea una terminal virtual con una serie de características que no permiten que en nuestro flujo usemos programas o transacciones que hagan SIGNON o SIGNOFF como la CESN. Si ejecutamos algún programa que utilice estos comandos, la invocación del flujo fallará

Aparte de estos dos comandos hay otros que no se pueden utilizar en un flujo de tipo Link3270 Bridge. Si tenemos dudas sobre si podemos hacer uso de un programa en nuestro flujo o si este va a hacer que se produzca un error al invocarlo, podemos hacer uso de la utilidad Load module scanner, DFHEISUP, usando la tabla DFHEIDBR. Si pasamos esta utilidad sobre el data ser donde tengamos los módulos compilados listos para ser ejecutados, los Load Modules, este programa nos lista aquellos comandos que sean EXEC CICS. La tabla DFHEIDBR filtra aquellos comandos que van a producir un error si los usamos por medio del Link3270 Bidge. En definitiva, los módulos donde nos indique que ha encontrado estos comandos son módulos que no vamos a poder utilizar para grabar nuestros flujos si queremos utilizar un flujo de tipo Link3270 Bridge ya que van a producir, o pueden producir, errores.

En caso de querer usar transacciones que requieran estar identificados para usarlas, haremos uso de la seguridad del CTG ya que al lanzar una petición con TPN, esta se ejecuta como si el usuario que la lanza se hubiese identificado en el sistema.

A continuación podemos ver una lista con los comandos contenidos en la tabla DFHEIDBR. Vemos como al final de la misma se encuentran los comandos SIGNON y SIGNOFF que, como ya habíamos indicado, no pueden ser usados.

- EXTRACT LOGONMSG
- ISSUE PRINT
- RECEIVE MAP INPARTN *
- SEND * DEFRESP
- SEND CONTROL ACTPARTN *
- SEND CONTROL OUTPARTN *
- SEND MAP ACTPARTN *
- SEND MAP OUTPARTN *
- SEND TEXT ACTPARTN *
- SEND TEXT OUTPARTN *
- ROUTE *
- SEND CONTROL MSR *
- SEND CONTROL LDC *
- SEND CONTROL OUTPARTN *
- SEND CONTROL ACTPARTN *
- SEND CONTROL PAGING *
- SEND CONTROL REQID *

- SEND CONTROL HONEOM *
- SEND CONTROL L40 *
- SEND CONTROL L64 *
- SEND CONTROL L80 *
- SEND MAP NLEOM *
- SEND MAP FMHPARM *
- SEND MAP LDC *
- SEND MAP PAGING *
- SEND MAP REQID *
- SEND MAP NOFLUSH *
- SEND MAP HONEOM *
- SEND MAP L40 *
- SEND MAP L64 *
- SEND MAP L80 *
- SEND MAP MAPPINGDEV ACCUM *
- SEND TEXT NLEOM *
- SEND TEXT FMHPARM *
- SEND TEXT LDC *
- SEND TEXT PAGING *
- SEND TEXT REQID *
- SEND TEXT HONEOM *
- SEND TEXT L40 *
- SEND TEXT L64 *
- SEND TEXT L80 *
- SEND TEXT JUSTIFY *
- SIGNON *
- SIGNOFF

Se podría usar FEPI como tipo de flujo para resolver algunos de estos problemas pero la infraestructura necesaria para que funcionen este tipo de flujos no está disponible en nuestro sistema.

3.4.- Invocación de flujos

Una vez que hemos instalado el SFF y creado e instalado nuestro primer flujo, vamos a ver como hacer peticiones a estos flujos. Para ello vamos a ver como hacerlo por medio del CTG o por medio de servicios Web. Principalmente nos interesa el primer método aunque vamos a ver el segundo ya que puede ser útil en algunos casos.

3.4.1.- Invocación por medio del CTG

A la hora de invocar flujos desplegados en el CICS por medio del CTG tenemos la posibilidad de hacerlo por medio de una interfaz de COMMAREA o de una interfaz de Canales y Contenedores. Además de las ventajas que hemos visto que presenta esta segunda interfaz, desde IBM se recomienda usarla frente a la interfaz de COMMAREA por lo que es la que vamos a usar para invocar flujos.

Para invocar un flujo por medio del CTG tenemos que hacer una petición ECI al programa DFHMADPL ya que es el programa del SFR que hace de interfaz para realizar

peticiones. A este programa le pasamos dos Contenedores, obligatoriamente con los nombres DFHMAC-REQUESTV1 y DFHMAC-USERDATA.

En el primero le pasamos el "Nombre de petición" que hemos definido en el archivo de propiedades de generación que es el mismo que el nombre del recurso PROCESSTYPE. Esto hará que se ejecute el flujo asociado. El nombre del PROCESSTYPE que le pasemos tiene que ser obligatoriamente de 8 caracteres. Si el nombre que le hemos dado nosotros es de menor longitud, rellenamos con espacios en blanco hasta llegar a los 8 caracteres.

En el segundo le pasamos los datos de entrada que necesite el flujo para ejecutarse. Si tenemos más de una variable de entrada, como es este caso, introducimos las dos seguidas en el Contenedor sin dejar espacios entre ellas. Por ejemplo, en este caso para pasarle la "accion" 2 y el "articulo" 0030 introducimos en el Contenedor la cadena "20030".Si la longitud de los datos que le pasamos para una variable es menor que la longitud total de la variable, por ejemplo porque le podemos pasar un nombre de 50 caracteres pero el que le pasamos es de sólo 8, tenemos que rellenar el reto de la variable hasta su longitud máxima, en este caso 42 caracteres, con espacios en blanco.

El programa DFHMADPL nos puede devolver dos Contenedores, el DFHMAC-ERROR y el DFHMAC-USERDATA.

En el primero nos va a informar de cualquier error que se haya producido al ejecutar el flujo. Si el programa nos devuelve este Contenedor es porque se ha podido producir un error y en ese caso la información que contenga el Contenedor DFHMAC-USERDATA no tiene por que ser válida.

Si el Contenedor de error no existe quiere decir que no se han producido errores al ejecutar el flujo, aunque se hayan podido producir al lanzar la petición por la conexión con el CTG, etc. Si tampoco se han producido errores de este tipo, que controlaremos por medio del ReturnCode que devuelve la petición ECI, en este Contenedor nos vamos a encontrar con los datos de salida del flujo.

En el Documento 3.3 que tenemos a continuación vamos a ver un ejemplo de cómo hacer una petición del flujo que acabamos de crear desde una aplicación Java por medio del CTG.

Como vamos a usar el CTG, lo primero que haremos será importar al proyecto la librería ctgclient.jar y crear en el main de la aplicación una JavaGateway.

Veremos como el constructor de la clase ECIRequest cambia y en vez de pasarle una array de bytes que simule una COMMAREA y su longitud, le pasamos una clase Channel con sus Containers

```
//Variable para dar un nombre al Canal que le vamos a pasar al DFHMADPL
//Puede ser cualquier nombre.
String CHANNEL = "CTGCHANNEL";

//Variable para pasar el identificador del flujo que queremos invocar
//Tiene que ser siempre de longitud 8 (de ahí los espacios en blanco)
String flujo = "EGUI1PR ";

////Variable para pasar los datos que necesite el flujo
String datos = "20030";

//Variable para crear mensajes de salida
String salida = "";

//Metemos toda la ejecución dentro de un bloque try/catch para poder capturar
//posibles excepciones que se produzcan
try {
```

```
	//Creamos una JavaGateway con la URL y el Puerto donde está escuchando
	//el CTG
	JavaGateway javaGateObject = new JavaGateway("20.20.20.1", 2006);

	//Creamos un Nuevo Channel con el nombre que hemos indicado en la
	//variable correspondiente
	Channel theChannel = new Channel(CHANNEL);

	//Creamos un Container dentro del Channel donde le pasamos el nombre
	//del PROCESSTYPE que queremos invocar
	theChannel.createContainer("DFHMAC-REQUESTV1", flujo);

	//Creamos un Container para pasarle posibles datos que necesite el
	//flujo
	theChannel.createContainer("DFHMAC-USERDATA", datos);

	//Creamos una ECIRequest para hacer la petición
	ECIRequest eciRequestObject = new ECIRequest(
		ECIRequest.ECI_SYNC_TPN,
		"CICS1",
		"mikel",
		null,
		"DFHMADPL",
		null,
		theChannel,
		ECIRequest.ECI_NO_EXTEND,
		0
	);

	//Lanzamos la petición
	javaGateObject.flow(eciRequestObject);

	//Comprobamos que no se haya producido ningún error al lanzar la
	//petición (Return Code = 0)
	if (eciRequestObject.getRc() == 0) {

		//Bloque try/catch para comprobar si existe el Contenedor de
		//error DFHMAC-ERROR que hemos dicho antes
		try {

			//Obtenemos el contenedor de error
			Container errorcont = theChannel
					.getContainer("DFHMAC-ERROR");

			//Comprobamos si el Contenedor de error es de tipo BIT o
			//de tipo CHAR
			switch (errorcont.getType()) {

				//Contenedor de tipo CHAR
				case CHAR:

//Creamos un mensaje de error con la infamación que consideramos oportuna y
//lo mostramos por pantalla
					salida ="ERROR: CICS Returned container["
						+ errorcont.getName() + "] "
						+ " Data["
						+ errorcont.getCHARData().trim()
						+ "]\n";

					System.out.print(salida);
```

```
                              //Hacemos un break para salir del switch
                              break;

                        //Contenedor de tipo bit
                        case BIT:

//Obtenemos el Contenedor de error, creamos un mensaje con la información
//que consideremos necesaria y lo mostramos por pantalla
                              byte[] data = errorcont.getBITData();

                              salida ="ERROR: CICS Returned container["
                                    + errorcont.getName() + "] "
                                    + " Data["
                                    + new String(data, "IBM037)")
                                    + "]\n";

                              System.out.print(salida);
                  }

//Capturamos la excepción que se produce cuando no hay Contenedor de error
            }catch (ContainerNotFoundException e) {

//Al no haber Contenedor de error, consideramos que la ejecución es correcta
//Creamos un mensaje para indicar que la ejecución ha sido correcta
                  salida = "Llamada ECI finalizada correctamente, el Canal
                        contiene "
                        + theChannel.getContainers().size()
                        + " contenedores\n";

                  System.out.print(salida); //Mostramos el mensaje

//Una vez que hemos comprobado que no hay Contenedor de Error tratamos el
//Contenedor con los datos de salida
//Bloque try/catch para tratar el Contenedor con los datos de salida
                  try {

                        //Obtenemos el Contenedor con los datos de salida
                        Container cont = theChannel
                            .getContainer("DFHMAC-USERDATA");

                        //Comprobamos el tipo de contenedor
                        switch (cont.getType()) {

                              //Contenedor de tipo CHAR
                              case CHAR:

                                    //Cogemos los datos que nos devuelve el
                                    //flujo
                                    String str2 = cont.getCHARData();

//Creamos un mensaje de salida con los datos que creamos oportunos
                                    salida = "USERDATA: Contenedor CICS de
      Retorno["
                                    + cont.getName() + "] Tipo["
                                    + cont.getType() + "] CCSID["
                                    + cont.getCCSID() + "] Longitud["
                                    + str2.length() + "]";

                                    System.out.println(salida); //Mostramos
      el mensaje de salida
```

```
//Mostramos los resultados que hemos obtenido
                                    salida = "Resultado\n" + str2 + "\n";
                                    System.out.println(salida);

                                    //Hacemos un break para salir del switch
                                    break;

                               //Contenedor de tipo BIT
                               case BIT:

                                    //Cogemos los datos que nos devuelve el
                                    //flujo
                                    byte[] data = cont.getBITData();

//Creamos un mensaje de salida con los datos que creamos oportunos
                                    salida = "USERDATA: Contenedor CICS de
      Retorno["
                                    + cont.getName() + "] Tipo["
                                    + cont.getType() + "] CCSID["
                                    + cont.getCCSID() + "] Longitud["
                                    + data.length + "]";

                                    //Mostramos el mensaje
                                    System.out.println(salida);

//Mostramos los resultados que hemos obtenido
                                    str2 = new String(data, "IBM037");
                                    salida = "Resultado\n" + str2 + "\n";
                                    System.out.println(salida);
                          }

//Excepción en caso de no encontrarse el contenedor con los datos que
//devuelve el flujo
                  }catch (ContainerNotFoundException e) {

//Mostramos un mensaje informando del error
                          salida = "No hay datos de salida";
                          System.out.print(salida);

//Cualquier otra excepción al tratar el Contenedor de Salida
                  }catch (Exception e) {

//Mostramos un mensaje informando del error
                          salida = "ERROR: Excepción manejando el contenedor de
datos:"                             + e.toString() + "\n";

                          System.out.print(salida);
                  }

//Capturamos cualquier otra excepción que se pueda producir manejando el
//Contenedor de error
            }catch (Exception ex) {

//Creamos un mensaje de error para informar de la excepción
                  salida = "ERROR: Excepción manejando el Contenedor de
                          error:"
                          + ex.toString() + "\n";

                  System.out.print(salida);
            }
```

```
//Se ha producido un error al lanzar la ejecución (Return Code ≠ 0)
      } else {

//Creamos un mensaje con el error que se ha producido
            salida = " ECI request failed: RC = "
                  + eciRequestObject.getRcString() + "("
                  + eciRequestObject.getRc() + ")";

//El Return Code indica que se ha producido un ABEND
                  if (eciRequestObject.getRc() ==
ECIRequest.ECI_ERR_TRANSACTION_ABEND) {

//Añadimos al mensaje información del ABEND
                        salida = salida + " Abend=" +
eciRequestObject.Abend_Code;
                  }

                  System.out.println(salida); //Mostramos el mensaje de error
      }

                  //Cerramos la conexión con el CTG
                  javaGateObject.close();

//Se ha producido una excepción al establecer la conexión con el CTG
}catch (IOException ioe) {

                  //Creamos un mensaje con el error y lo mostramos
                  salida = salida + "No se puede conectar con el CTG: "
                        + ioe.toString() + "\n";

                  System.out.print(salida);

//Se ha producido cualquier otra excepción
}catch (Exception e) {

                  //Creamos un mensaje de error y lo mostramos
                  salida = salida + "Handled exception: "
                        + e.toString() + "\n";

                  System.out.print(salida);
}
```

Documento 3.3 Ejemplo de programa Java para llamar a flujo del SFF

De esta forma invocamos un flujo por medio del CTG. Para hacer el programa más flexible realmente desarrollaríamos una GUI donde recogeríamos ciertos datos que pueden cambiar como el nombre del flujo que queremos invocar, los datos de entrada y en algunos casos el nombre del CICS la IP del CTG, etc. Hay otros datos que tiene que ser siempre los mismo como el nombre del programa que queremos invocar.

El nombre del PROCESSTYPE que le pasamos en el Contenedor DFHMAC-REQUESTV1 tiene que ser en mayúsculas. En caso de introducirlo en minúsculas probablemente recibiremos el Contenedor DFHMAC-ERROR diciéndonos que se ha producido el error DFHMA06021E que indica que el nombre no es correcto o que el PROCESSTYPE no está instalado. Esto es debido a que el nombre del recurso está definido en el CICS con mayúsculas.

Los Contenedores que creamos para pasar el nombre del PROCESSTYPE y los datos de entrada pueden ser tanto de tipo CHAR como BIT. La única diferencia es que si enviamos los datos de entrada en un Contender de tipo BIT los datos de salida también los recibiremos en tipo

BIT y tendremos que convertir tanto los datos de entrada como de salida con la página de códigos IBM037 que usa el CTG.

Si hacemos que el flujo llegue a una pantalla que no hemos grabado, por ejemplo porque introducimos la "accion" 1, no se producirá un error aunque no obtendremos ningún resultado ni en el Contendor de salida ni en el de error.

En caso de haber definido más de una variable de salida, recibiremos ambas en el Contenedor DFHMAC-USERDATA una seguida de la otra, como si fuese una COMMAREA. Este es un pequeño inconveniente a la hora de mostrar los datos de salida ya que tenemos que saber la longitud exacta de cada variable para poder trocear la salida. Este inconveniente se espera que se solucione con futuras versiones del SFF.

De esta forma ya hemos visto como invocar flujos por medio del CTG.

3.4.2.- Invocación por medio de servicios Web

El CICS puede actuar como servidor de servicios Web y nos permite invocar los flujos del SFF haciendo una petición a un servicio Web por ejemplo desde una aplicación Java o .Net o de cualquier otro método. Nosotros vamos a ver como hacerlo desde una aplicación de escritorio escrita en cualquiera de estos dos lenguajes.

Aunque el CICS puede actuar como servidor de servicios Web no ofrece los servicios de registro y publicación de los mismos. Esta tarea la realiza por ejemplo el producto *WebSphere Registry and Repository* para el que no tenemos licencia. Es por esto que tenemos que transferir por FTP en modo ASCII el archivo WSDL del mainframe a la máquina donde queremos desarrollar y utilizar la aplicación e importarlo en todas las aplicaciones. Esto hace que sea un poco engorroso el invocar flujos de esta forma aunque nos sirve para hacernos una idea de los pasos a seguir.

Para poder hacer uso del CICS como servidor de servicios Web, tenemos que crear un recurso TCPIPSERVICE para que escuche en un puerto las peticiones que le lleguen. Este recurso lo definiremos en el puerto 13311. Este puerto es el que hemos definido en el archivo de propiedades de generación para escuchar peticiones para el servicio Web.

Para definir el recurso nos conectamos el CICS y nos identificamos por medio de la transacción CESN. Al TCPIPSERVICE le daremos el nombre SFFTCPS y lo definiremos en el grupo CICSSFRG que es donde están todas las definiciones de recursos para el SFR. Para ello, una vez identificados en el CICS ejecutamos el comando CEDA DEF TCPIPSERVICE(SFFTCPS) G(CICSSFRG). Nos aparecerá una pantalla similar a la de la Figura 2.20 cuando definimos los recursos necesarios para las conexiones del CTG al CICS.

La mayoría de los atributos los dejaremos con el valor por defecto excepto las siguientes:

- POrtnumber : 13311
- Maxdatalen : 000032

Una vez que tenemos definido el TCPIPSERVICE e instalado ya podemos realizar peticiones de servicios Web al CICS para que invoquen un flujo.

Como ya hemos dicho, el primer paso antes de poder crear la aplicación que llame al servicio Web es transferir a nuestra máquina el archivo WSDL. Esto lo hacemos por medio del BlueZone en modo ASCII. El archivo WSDL se encontrará en la ruta "/repositorio/wsdir/EGUI/prueba/" si hemos seguido todos los pasos que hemos ido haciendo. Esta ruta es la que hemos indicado a la hora de generar los archivos del flujo.

Una vez que hemos transferido el archivo WSDL a nuestra máquina ya podemos empezar a desarrollar la aplicación. Vamos a ver como desarrollar una aplicación de escritorio tanto Java como VB.Net para llamar al servicio Web y que este invoque al flujo.

3.4.2.1.- Llamar a servicios Web desde Java

Para poder llamar a un servicio Web desde Java, el primer paso es importar el archivo WSDL asociado al servicio Web a la aplicación que va a llamar al servicio Web. Para ello creamos un nuevo proyecto Java. Hacemos clic derecho sobre el proyecto que queremos que llame al WebService y elegimos "Nuevo → Web Service Client…" como podemos ver en la Figura 3.87.

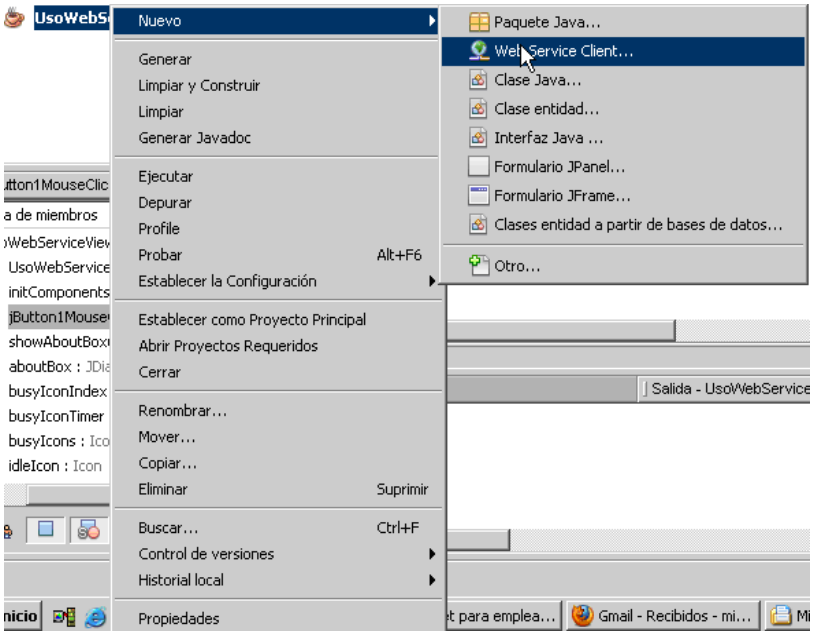


Figura 3.87 Añadir cliente de servicio Web a aplicación Java

Nos aparecerá la pantalla de la Figura 3.88 donde podemos elegir diferentes maneras de cargar el servicio Web. Podemos cargarlo desde un proyecto que hayamos creado anteriormente, desde un archivo local, como es nuestro caso, o teniendo el WSDL en una URL, por ejemplo subido a un *WebSphere Registry and Repository*.

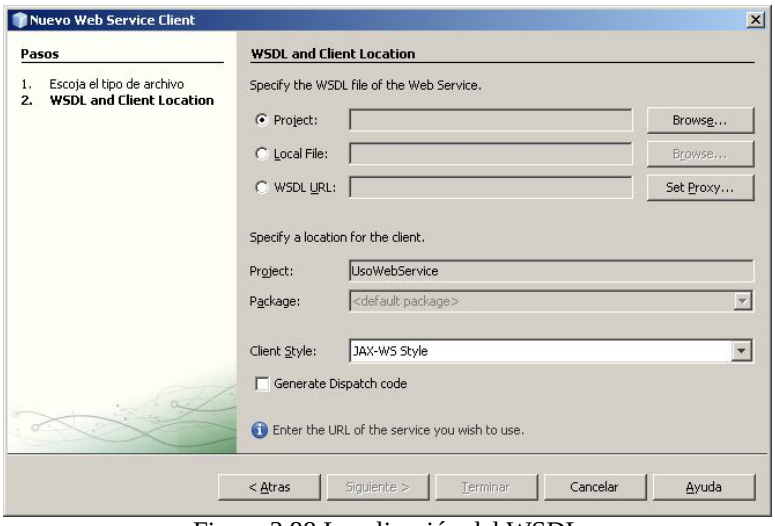


Figura 3.88 Localización del WSDL

En nuestro caso elegimos "Local File" y pulsamos el botón "Browse…". Nos aparecerá una ventana donde buscaremos el archivo WSDL que nos hemos descargado previamente del mainframe.

Una vez hecho esto nos dirigimos al sitio del código Java desde donde queremos hacer la llamada al servicio web. Hacemos clic derecho y elegimos "Web Service Client Resources → Call Web Service Operation…" como vemos en la Figura 3.89.

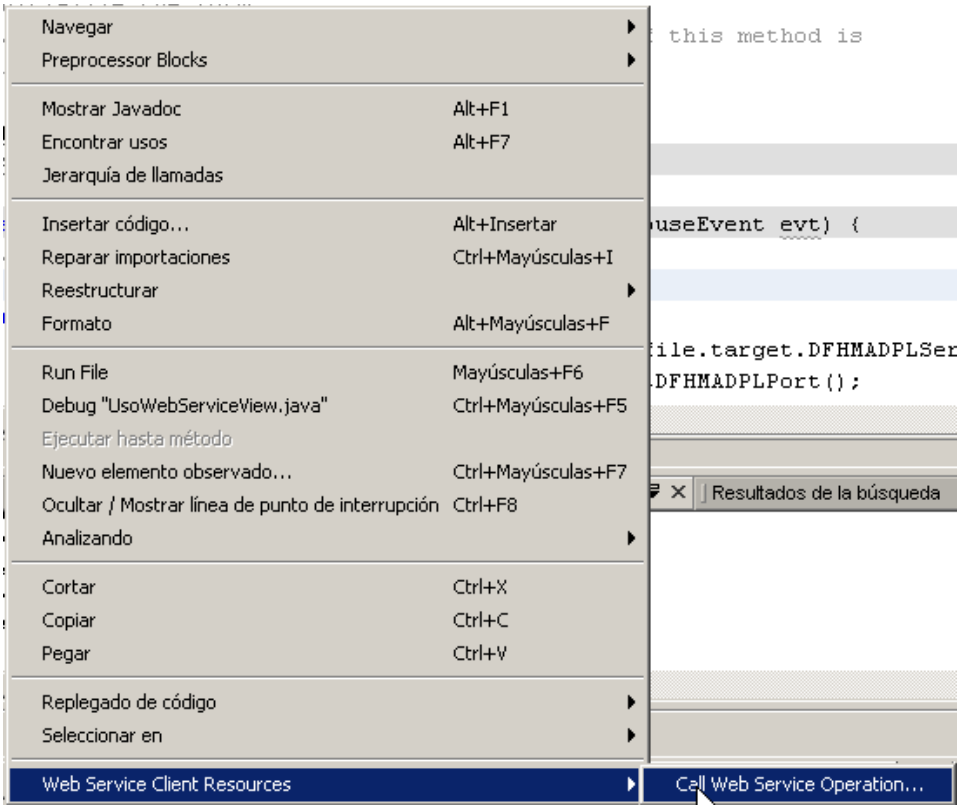


Figura 3.89 Añadir llamada a servicio Web

Nos aparecerá la pantalla de la Figura 3.90 donde, en caso de tener más de un servicio Web en nuestra aplicación, o más de una operación en un servicio Web, elegir el servicio Web y operación a la que llamar. Como nosotros sólo tenemos un servicio Web con una operación la elegimos.

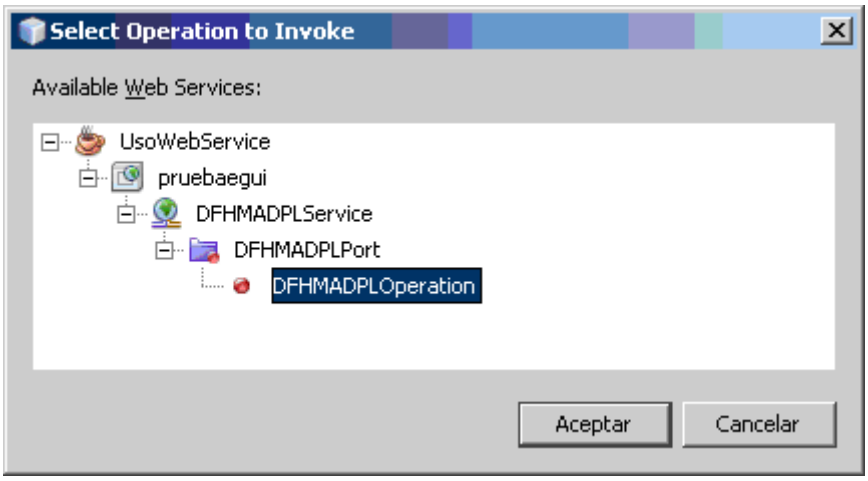


Figura 3.90 Elegir servicio Web y operación

Nos aparecerá código Java insertado. En la cabecera del documento Java desde el que queremos llamar al servicio Web tendremos que importar dos interfaces de entrada y salida. Todo esto lo podemos ver en el Documento 3.4.

```
//Importamos en la cabecera del documento Java desde el que vamos a llamar al
//servicio Web las siguientes interfaces de entrada y salida
import com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.*;
import com.xmlenablo.schemas.xmlenablointerface.ProgramInterface.*;


//Código situado en el punto del código desde el que queremos llamar al
//servicio Web
//Bloque try para hacer la invocación de la operación
try {

      //Creamos las clases necesarias para poder hacer la invocación
      file.target.DFHMADPLService service = new
file.target.DFHMADPLService();
      file.target.DFHMADPLPort port = service.getDFHMADPLPort();

      //Creamos la interfaz necesaria para poder hacer la invocación de la
operación
      com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.ServiceInputA
rea serviceInputArea = new
com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.ServiceInputArea();
      com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.ServiceInputA
rea.DfhmahrRequestv1 dfhmahrRequestv1 = new
com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.ServiceInputArea.Df
hmahrRequestv1();

      //Le decimos el nombre del flujo que queremos que se ejecute
      dfhmahrRequestv1.setDfhmahrRequestname("EGUI1PR");
      serviceInputArea.setDfhmahrRequestv1(dfhmahrRequestv1);

      //Modificamos la interfaz para añadirle los datos necesarios para
efectuar la invocación
      com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.ServiceInputA
rea.InputDataContainer inputDataContainer = new
com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.ServiceInputArea.In
putDataContainer();
      com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.ServiceInputA
rea.InputDataContainer.IDfh0Xs1ExmenuExmenu iDfh0Xs1ExmenuExmenu = new
com.xmlenabli.schemas.xmlenabliinterface.ProgramInterface.ServiceInputArea.In
putDataContainer.IDfh0Xs1ExmenuExmenu();

      //Tendremos un setXXX por cada uno de las variables de entrada que
      //hayamos definido en el flujo
      iDfh0Xs1ExmenuExmenu.setaccion("2");
      iDfh0Xs1ExmenuExmenu.setarticulo("0030");
      inputDataContainer.setIDfh0Xs1ExmenuExmenu(iDfh0Xs1ExmenuExmenu);
      serviceInputArea.setInputDataContainer(inputDataContainer);

      //Llamamos a la operación y recogemos su salida
      com.xmlenablo.schemas.xmlenablointerface.ProgramInterface.ServiceOutput
Area result = new
com.xmlenablo.schemas.xmlenablointerface.ProgramInterface.ServiceOutputArea()
;
      result = port.dfhmadplOperation(serviceInputArea);
```

```
      com.xmlenablo.schemas.xmlenablointerface.ProgramInterface.ServiceOutput
Area.NavigatorOutputArea navigatorOutputArea =
result.getNavigatorOutputArea();
      com.xmlenablo.schemas.xmlenablointerface.ProgramInterface.ServiceOutput
Area.NavigatorOutputArea.ODfh0Xs1ExmenuExmenu oDfh0Xs1ExmenuExmenu =
navigatorOutputArea.getODfh0Xs1ExmenuExmenu();

      //Recogemos el número de filas de salida que nos ha devuelto el
      //servicio Web
      int i;
      i = oDfh0Xs1ExmenuExmenu.getsalida().size();

      //Recorremos todas las filas que nos ha devuelto el servicio Web
      //y las mostramos
      int j = 0;
      System.out.println("Resultado obtenido de la llamada al servicio Web");
      for (j=0;j < i; j++){
            System.out.println(oDfh0Xs1ExmenuExmenu.getSalida().get(j).getSal
ida());
      }

//Bloque catch para capturar las excepciones que se puedan producir al
//invocar la operación
} catch (Exception ex) {

      //Mensajes que mostramos en caso de que se produzca una excepción
      System.out.println("Excepcion al llamar al servicio Web");
      System.out.println(ex.getMessage())
      System.out.println(ex.toString());
}
```

Documento 3.4 Llamar a un servicio Web desde Java

Vemos como al usar servicios Web tenemos una entrada por cada una de las variables de entrada que hayamos definido. También tendríamos separadas cada una de las variables de salida y, en caso de haber definido la variable de salida como una matriz, cada fila de la matriz separada.

Esto es una ventaja con respecto a las llamadas desde Canales y Contenedores donde todas las entradas y todas las salidas eran un flujo de caracteres continuo.

El inconveniente es tener que transferir el archivo WSDL a cada máquina e importarlo en cada aplicación que quiera hacer uso del servicio Web.

3.4.2.2.- Llamar a servicios Web desde VB.Net

Los pasos para llamar a un servicio Web desde una aplicación VB.Net son similares a los seguidos desde Java. Primero tenemos que importar en la aplicación el archivo WSDL y luego escribir el código necesario para hacer la llamada.

Para importar el archivo WSDL, una vez que hemos creado el proyecto para la aplicación hacemos clic derecho en el Explorador de Soluciones sobre nuestro proyecto y elegimos “Agregar referencia Web…” como podemos ver en la Figura 3.91.

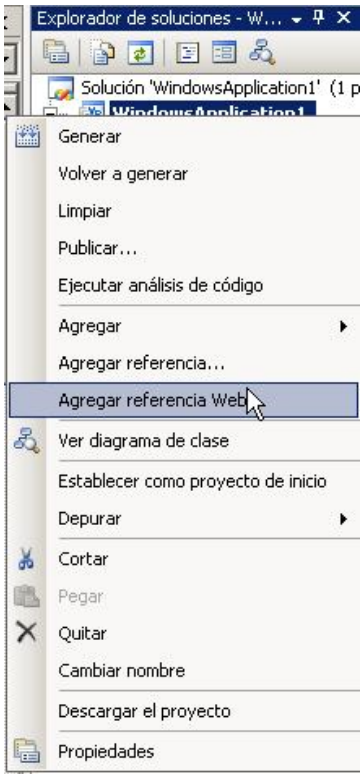


Figura 3.91 Importar archivo WSDL a aplicación .Net

Nos aparecerá la ventana de la Figura 3.92 donde indicarle la URL donde está el WSDL que queremos utilizar. Esta URL puede ser la ruta de nuestra máquina donde tenemos el archivo Web. Suponiendo que el archivo WSDl lo hemos almacenaddo en C:\Pruebas escribimos la ruta del WSDL y cuando le damos a "Ir" aparece las operaciones disponibles en ese WSDL. Ahora le podemos dar un nombre de referencia para poder usarlo más adelante, en nuestro caso le llamamos WebService ya que sólo vamos a tener uno, y le damos a "Agregar referencia" y con lo que podemos cerrar esta ventana.

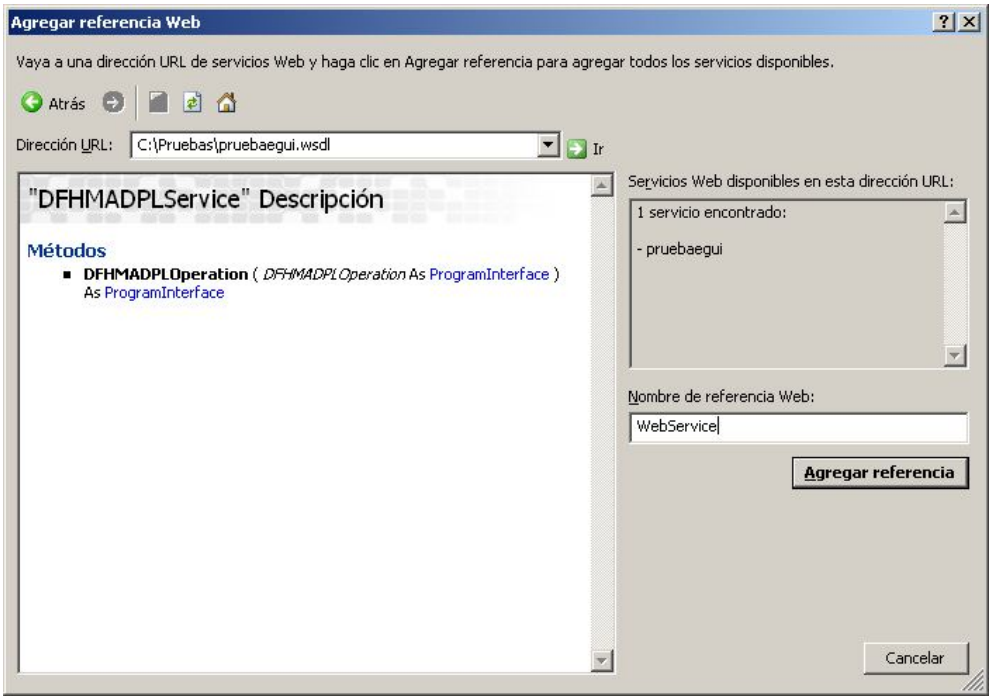


Figura 3.92 Agregar referencia Web a la aplicación

Una vez que tenemos el servicio Web añadido, ya podemos hacer uso de sus operaciones.

En el punto del código de la aplicación desde donde queremos llamar al servicio Web añadimos las siguientes líneas que podemos ver en el Documento 3.5.

```
‘Definimos una variable para hacer uso de las operaciones del servicio Web
Dim ws As New WebService.DFHMADPLService

‘Definimos una variable para dar nombre al flujo que queremos llamar
Dim flujo As New String("EGUI1PR")

‘Bloque Try/Catch para llamar a la operación del servicio Web
Try

      ‘Definimos una variable como interfaz de entrada para llamar a la
      ‘operación del servicio Web
      Dim pi As New WebService.ProgramInterface

      ‘Inicializamos las diferentes propiedades de la Interfaz que
      ‘necesitamos para invocar al WebService
      pi.service_input_area = New
WebService.ProgramInterfaceService_input_area

      pi.service_input_area.dfhmahr_requestv1 = New
WebService.ProgramInterfaceService_input_areaDfhmahr_requestv1

      ‘En requestname le decimos el nombre del flujo al que queremos invocar
      pi.service_input_area.dfhmahr_requestv1.dfhmahr_requestname = flujo

      ‘En input_data_container damos el valor de las variables de entrada
      ‘Tendríamos una por cada variable de entrada que hayamos definido
      pi.service_input_area.input_data_container.i_pruebaegui.accion = "1"
      pi.service_input_area.input_data_container.i_pruebaegui.articulo =
"0030"

      ‘Hacemos lo mismo de antes pero ahora para la interfaz de salida
      Dim pi1 As New WebService.ProgramInterface1
      pi1.service_output_area = New
WebService.ProgramInterfaceService_output_area
      pi1.service_output_area.navigator_output_area = New
WebService.ProgramInterfaceService_output_areaNavigator_output_area


      ‘Llamamos a la operación del servicio Web pasándole como entrada
      ‘la interfaz de entrada que hemos creado y guardando los resultados que
      ‘nos devuelve en la interfaz de salida
      pi1 = ws.DFHMADPLOperation(pi)


      ‘Una vez que nos ha contestado el servicio Web, obtenemos los datos que
      ‘nos han devuelto y los vamos recorriendo mostrandolos
      ‘En caso de tener más de una variable de salida, haríamos esto para
      ‘todas ellas
      Dim i As Integer
      i =
pi1.service_output_area.navigator_output_area.o_eguicompleto.salida.Length
      Dim j As Integer
      j = 0
      For j = 0 To i - 1
```

```
            Console.WriteLine(
pi1.service_output_area.navigator_output_area.o_pruebaegui.salida(j).ToString
)
      Next j

'Capturamos cualquier error que se haya podido producir
Catch ex As Exception

      Console.WriteLine(ex.Message)
      Console.WriteLine(ex.ToString)
End Try
```

Documento 3.4 Llamada a servicio Web desde VB.Net

Como podemos ver, al igual que la llamada desde Java tenemos las variables de entrada y de salida definidas por separado lo que facilita el tratamiento de los datos.

El inconveniente también es el mismo de tener que transferir el archivo WSDL a cada máquina e importarlo en cada aplicación que quiera hacer uso del servicio Web.

3.5.- Flujo con alternativas

Una vez que hemos visto como crear e instalar un flujo básico, y como invocarlo tanto por medio del CTG como desde un servicio Web, vamos a ver como desarrollar flujos más complejos.

Lo primero que vamos a ver es como crear un flujo con alternativas. Por ejemplo, para que en el flujo anterior permite introducir la acción 1 o la 2.

El primer paso que tenemos que hacer es grabar y guardar un flujo similar al que hemos grabado anteriormente. Una vez que hemos hecho esto, en la pantalla inicial de la transacción EGUI introducimos la acción 1 y pulsamos Control. Esto nos llevará a la pantalla de la Figura 3.93.

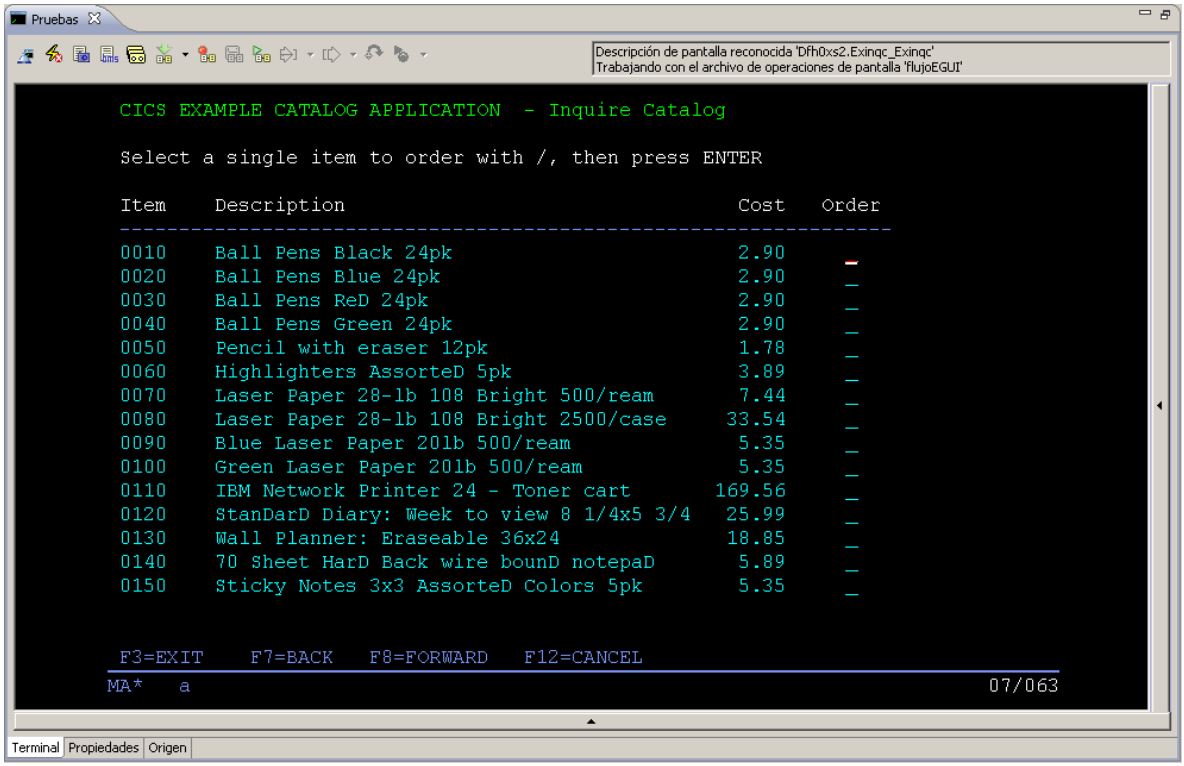


Figura 3.93 Pantalla de la acción 1 de la transacción EGUI

Una vez que estemos en esta pantalla cargamos el flujo que acabamos de grabar como hemos visto en el flujo básico para probarlo. Ahora pulsamos el botón que hemos visto en la Figura 3.31 para empezar a grabar un flujo. Nos aparecerá la ventana que vemos en la Figura 3.94 que nos da la posibilidad de añadir el flujo que vamos a grabar al flujo que tenemos cargado, o de grabarlo independientemente.

Figura 3.94 Añadir el flujo al cargado o crear uno nuevo

Pulsamos “Si” para añadir el nuevo flujo que vamos a grabar al flujo cargado. Nos aparecerá la ventana de la Figura 3.95 donde podemos grabar el nuevo flujo como un camino independiente o añadirlo a partir de un nodo o pantalla. Elegimos esta segunda opción y elegimos de la lista inferior la pantalla anterior a la que estamos para que se añada el nuevo flujo a partir de esta. En este caso como solo nos aparece una pantalla seleccionamos esta y pulsamos “Aceptar”.

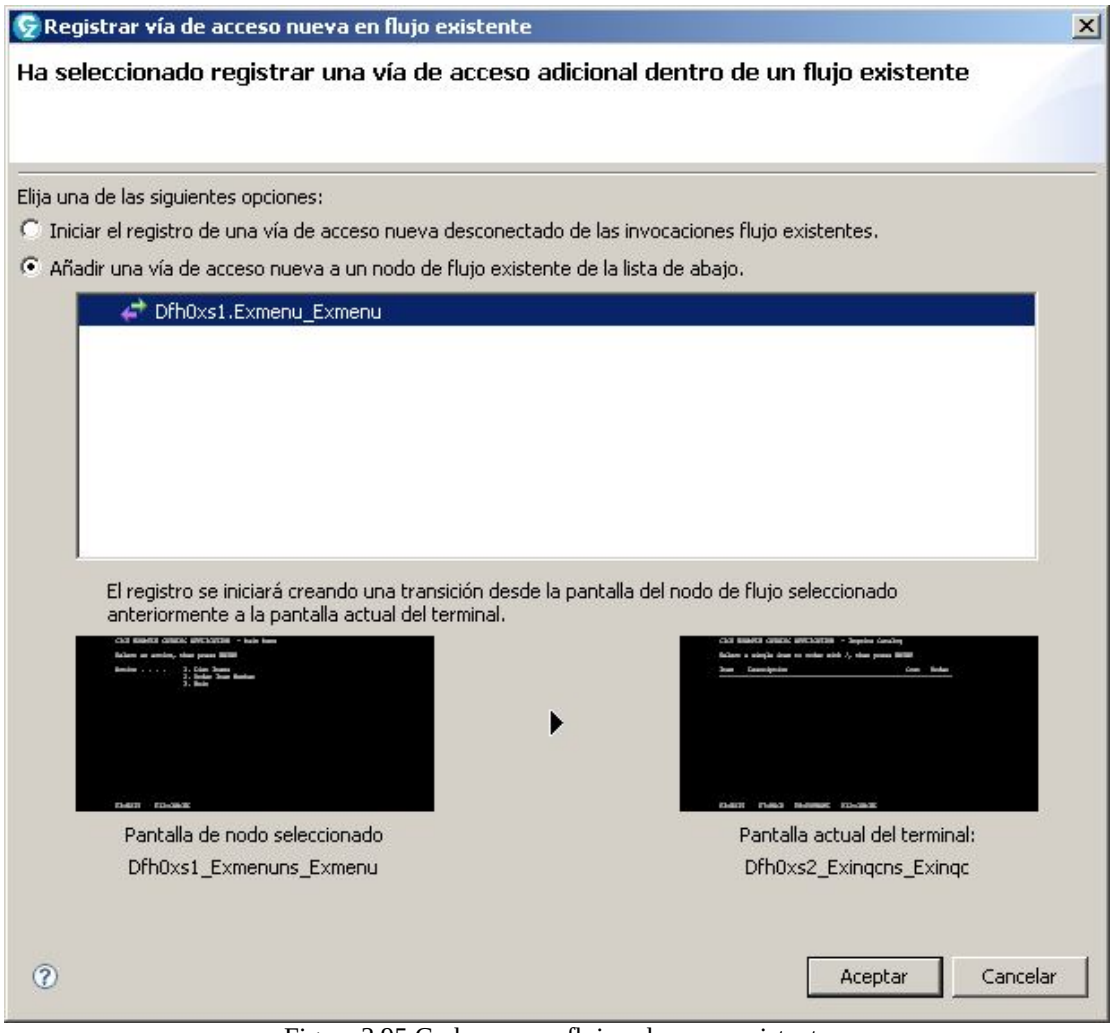


Figura 3.95 Grabar nuevo flujo sobre uno existente

Si nos fijamos en la parte inferior de la pantalla, vemos cual es el nodo seleccionado a partir del cual añadiremos el nuevo flujo y la pantalla en la que estamos que será la primera pantalla del nuevo camino.

Una vez hecho esto seguimos grabando el flujo de la forma normal. En este caso, lo único haremos será extraer los datos que aparecen en la pantalla de la Figura 3.93 en una variable de matriz de nombre "salida1". Ahora ya podemos finalizar la grabación del flujo y guardarlo con los botones de las Figuras 3.50 y 3.51.

Si ahora hacemos doble clic sobre el flujo, se abrirá el editor de flujos. En la Figura 3.96 podemos ver como del nodo de la pantalla inicial salen dos caminos, uno por cada uno de las alternativas que hemos grabado a partir de ese nodo.

Figura 3.96 Nodo inicial de la EGUI con dos alternativas

De esta forma podemos crear tantos caminos como queramos a partir de cualquier nodo. Por ejemplo, a partir del nodo inicial podríamos crear otro camino que simplemente se quede en la primera pantalla y recoja el mensaje que aparece cuando hemos introducido una acción o número de artículo inválido.

Esta técnica también nos sirve para extender con más pantallas un flujo que ya hayamos creado. Simplemente empezamos a grabar en la siguiente pantalla a la última que hemos grabado y unimos el nuevo flujo a la última pantalla que hemos grabado.

Una vez que hemos finalizado de grabar el flujo, podemos generar los archivos necesarios e instalarlo como hemos hecho anteriormente. Si probamos ahora el flujo por medio del RDz pasándole la "accion" 1 obtendremos en la variable "salida1" una línea por cada línea de la pantalla de salida como podemos ver en la Figura 3.97.

Figura 3.97 Salida de variable de tipo matriz

La variable “salida” estaría vacía.

Si hacemos la petición por medio del CTG, obtendríamos al principio una serie de espacios en blanco de la longitud de la variable “salida” y luego los datos contenidos en “salida1” de forma continua, sin obtener una línea por cada línea de salida. Obtenemos algo similar a lo que podemos ver a continuación entre comillas.

“ 0010 Ball Pens Black 24pk 2.900020 Ball Pens Blue 24pk 2.900030 Ball Pens ReD 24pk”

Al principio hay 76 espacios en blanco correspondientes a lo longitud de de la variable “salida” y a continuación una parte de los datos de la variable “salida1” sin separación entra la primera fila y la segunda, etc.

Aunque podamos crear un flujo con tantos caminos alternativos como queramos para realizar diferentes tareas, desde IBM se recomienda crear un flujo para cada camino ya que de esta forma es más fácil mantenerlos y comprobar donde se producen posibles errores.

3.6.- Flujo con bucles

Una vez que hemos visto como crear un flujo con múltiples caminos, vamos a ver cómo crear un flujo con bucles. Normalmente los bucles los vamos a utilizar para extraer los datos de un listado que ocupe más de una pantalla pero que no sepamos exactamente cuántas pantallas ocupa. Esto ocurre por ejemplo en el listado de artículos que se muestra con la acción 1 de la transacción EGUI. En un principio ocupa 2 pantallas pero si se agregasen más artículos no podemos saber cuántas pantallas va a ocupar.

Aunque esta es la mayor utilidad que le vamos a dar a los bucles también los podemos usar para otras cosas. Por ejemplo, si queremos extraer de una cuenta bancaria un saldo que sea múltiplo de 100, podríamos grabar un bucle que en cada iteración extraiga 100 del saldo total de la cuenta hasta que el saldo sea menor a esta cantidad. Este bucle probablemente usaría más de una pantalla diferente en cada iteración del bucle.

Vamos a utilizar la transacción EGUI para crear un flujo con un bucle que extraiga todos los datos del listado de artículos. El mismo método se podrá extrapolar a cualquier otro bucle que queramos grabar.

Para poder grabar un bucle sobre un listado se tienen que cumplir dos restricciones:

- Que el listado tenga al menos dos pantallas de datos
- Que haya una página distinta, o igual pero con un mensaje, que indique que hemos llegado al final del listado

La primera restricción es debido a que una iteración del bucle tiene que acabar en la misma pantalla que empezó. Por ello, al pulsar normalmente F8 para pasar a la siguiente pantalla del listado tiene que haber una segunda pantalla de listado similar a la anterior para poder decirle al RDz que eso es una iteración del bucle. Esto se verá más claramente cuando expliquemos como grabar el flujo. También veremos como en algunos casos esta restricción la podemos evitar.

La segunda restricción se debe a que necesitemos una pantalla diferente, ya sea porque es totalmente diferente o porque tiene un campo que indique el final del listado, que indique el final del listado. Esto lo necesitamos para decirle al flujo que cuando llegue a esta pantalla finalice el bucle. En la transacción EGUI, al llegar al final del listado se mostrará en un campo un mensaje que indique este hecho como podemos ver en la Figura 3.98.

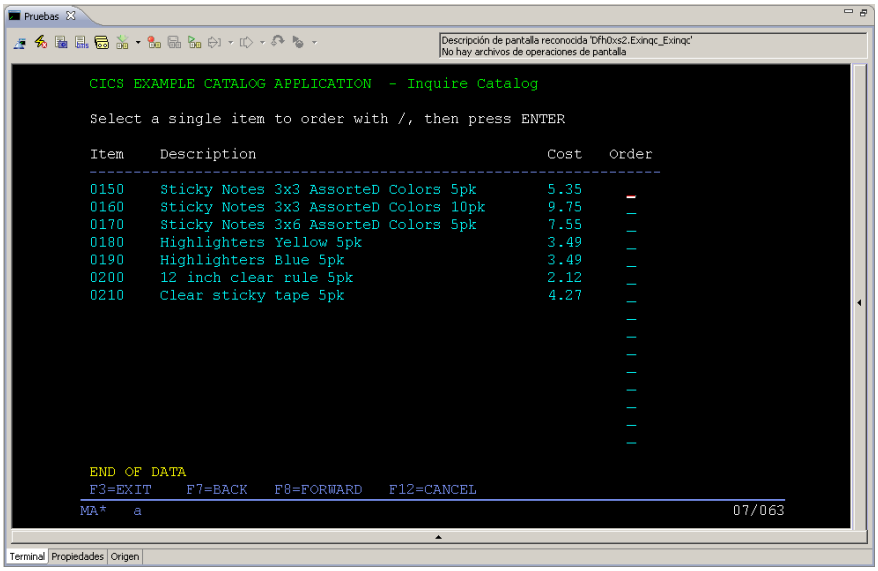


Figura 3.98 Pantalla de final de listado de la transacción EGUI

Por último, hay que indicar que estas pantallas de final de bucle no se tratan como se hace con el resto de pantallas del bucle. Por ejemplo, si en cada iteración del bucle extraemos todos los datos de la pantalla, los datos de la pantalla de la Figura 3.98 no se extraerían a no ser que lo hubiésemos hecho antes como es el caso, ya que primero aparece la pantalla con los datos y al volver a pulsar F8 aparece la misma pantalla pero con el mensaje de final de datos.

Antes de empezar a grabar el flujo tenemos que hacer ciertas modificaciones sobre la pantalla del listado para que el RDz diferencie cuando no tiene mensaje de final de datos y tiene que continuar el bucle de cuando si lo tiene y finalizar el bucle. Para ello, tenemos que saber cual es el nombre del mapa que representa esta pantalla que como podemos ver en la parte superior derecha de la Figura 3.98 es "Dfh0xs2.Exinqc.sfmxsd".

En la parte del "Explorador de proyectos" del RDz nos dirigimos a "pruebasEGUI → pruebasEGUI.Terminal → Mensajes". Aquí están los mapas que hemos importado o los que hemos podido capturar a la hora de grabar el flujo como podemos ver en la Figura 3.99. Si situamos el puntero sobre cada uno de las pantallas y lo mantenemos un momento sobre ella aparecerá una miniatura de la pantalla.

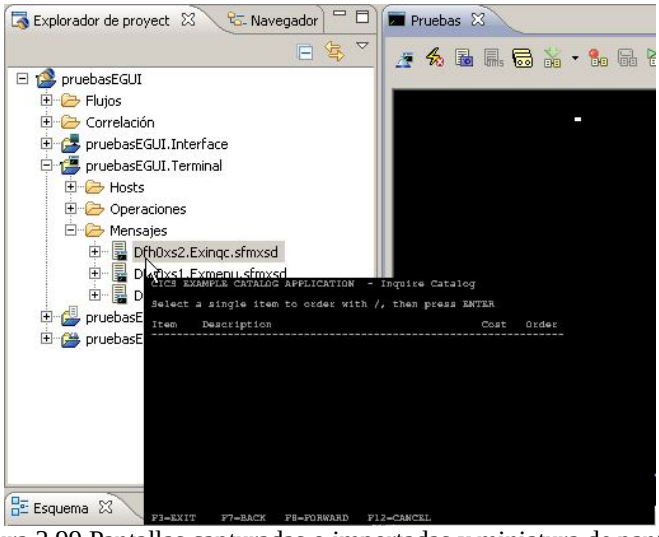


Figura 3.99 Pantallas capturadas o importadas y miniatura de pantalla

Hacemos doble clic sobre el mapa que hemos indicado para abrirlo en edición con lo que nos aparecerá en la parte derecha la pantalla de la Figura 3.100. Los campos de la pantalla marcados en azul son campos de entrada.

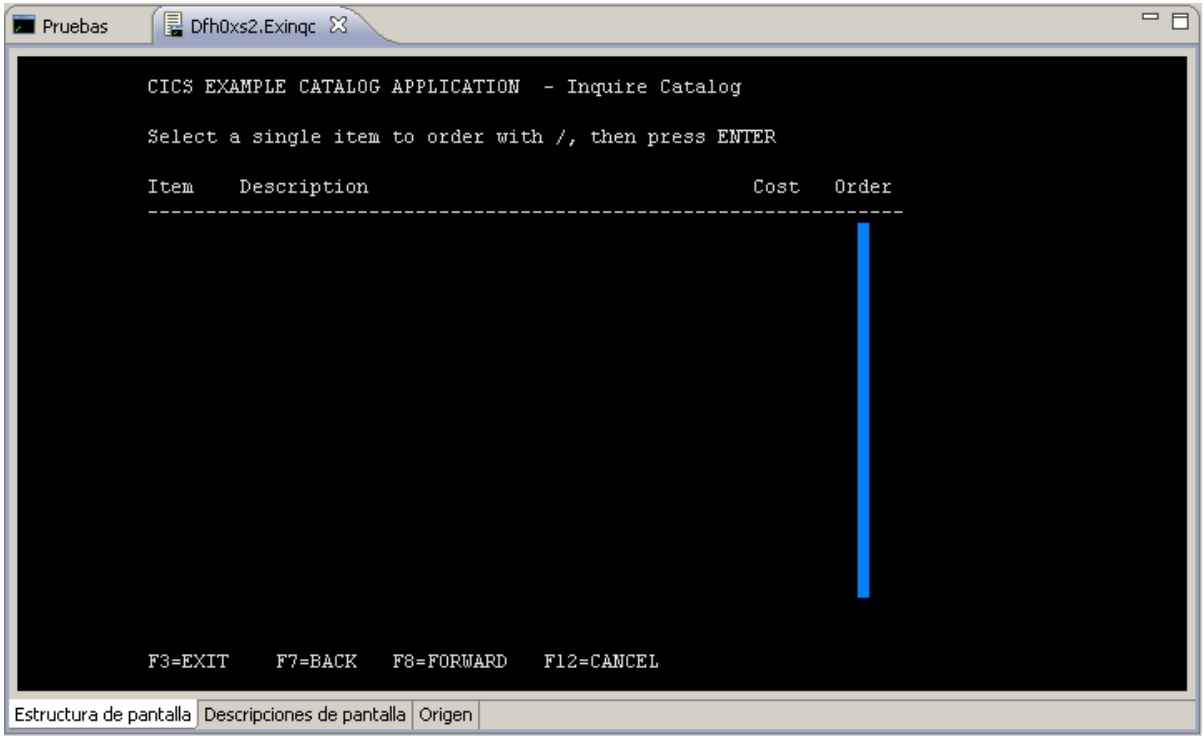


Figura 3.100 Pantalla abierta en edición

Pulsamos en la pestaña “Descripciones de pantalla” y se nos mostrará la pestaña de la Figura 3.101.

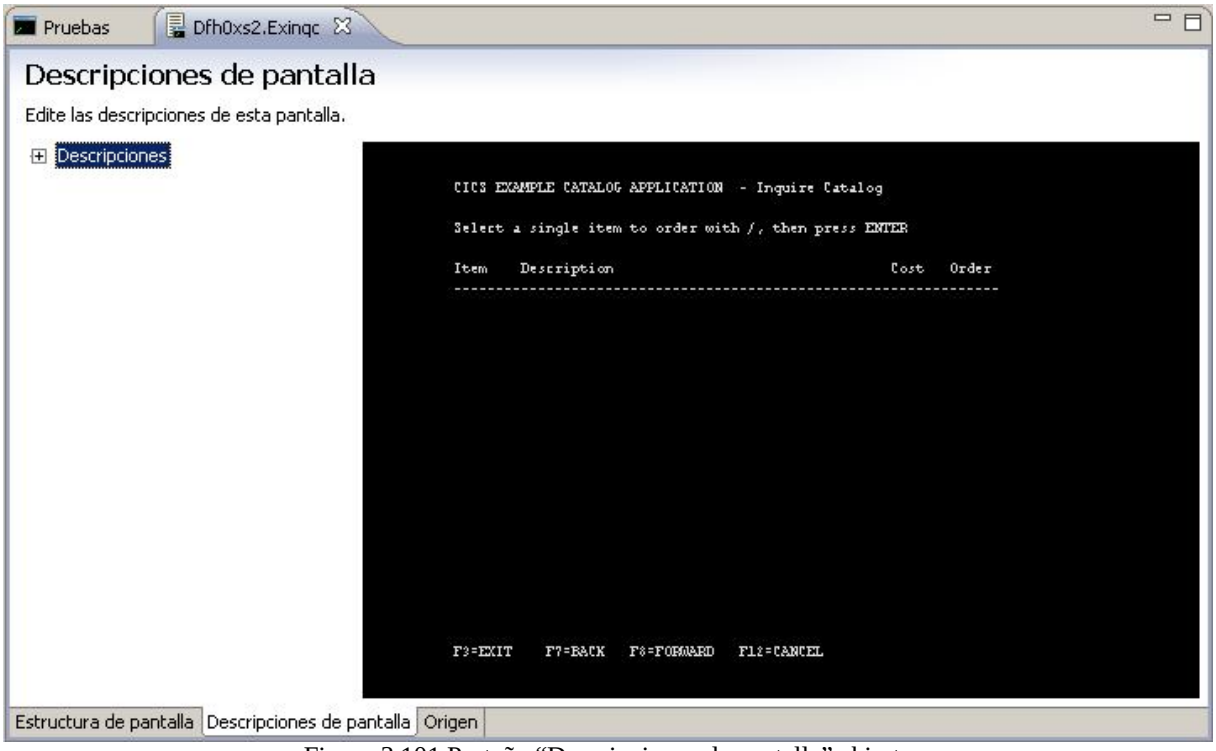


Figura 3.101 Pestaña “Descripciones de pantalla” abierta

Desplegamos el árbol “Descripciones” haciendo clic sobre la +, hacemos clic derecho sobre “Exinqc” y elegimos “Añadir descriptor → Añadir descriptor de patrón de campo” como podemos ver en la Figura 3.102.

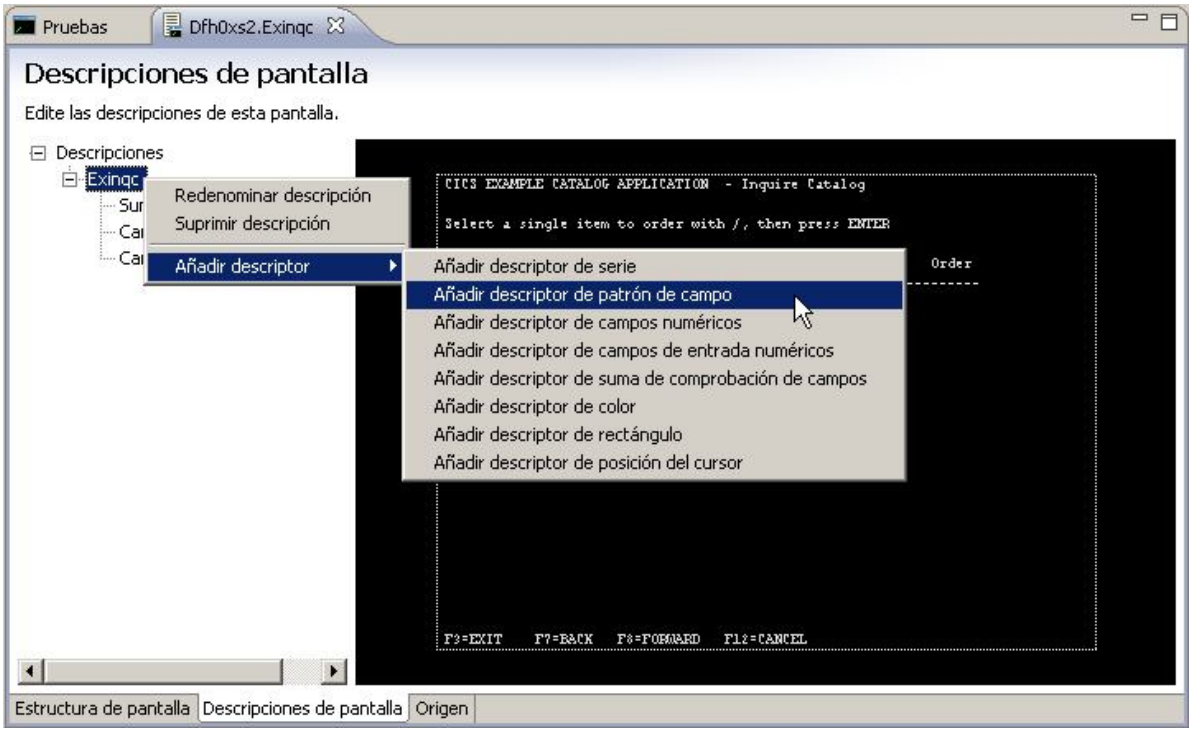


Figura 3.102 Añadir descriptor de patrón de campo

Veremos que nos aparece un recuadro rojo sobre el primer campo de la pantalla que arrastraremos sobre el campo en el que aparece el mensaje de final de datos que es el campo “INQC-MSG”. La pantalla quedará como podemos ver en la Figura 3.103.

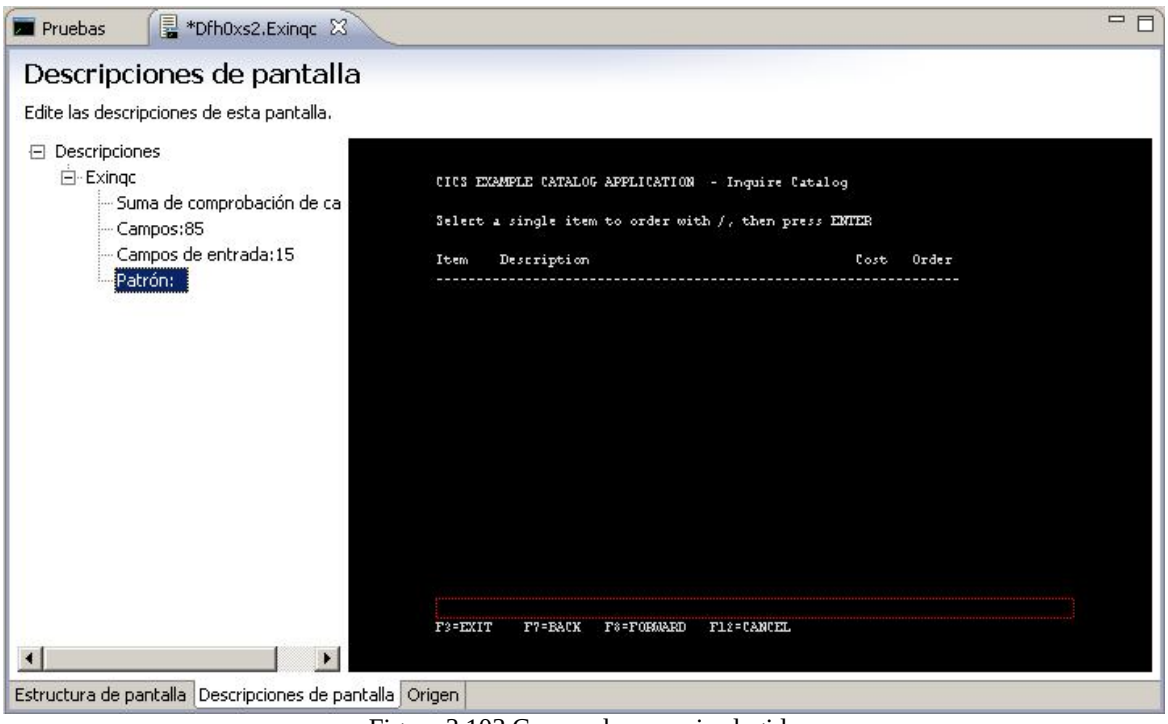


Figura 3.103 Campo de mensaje elegido

Esto nos permitiría diferenciar esta pantalla cuando tiene un mensaje de cuando no. En un principio esto nos valdría para grabar el bucle pero como más adelante vamos a necesitar diferenciar el mensaje de fin de datos de cualquier otro lo vamos a prepara en este momento. Para ello, seleccionamos el patrón y en la parte inferior de la pantalla del RDz seleccionamos la pestaña "Propiedades" como podemos ver en la Figura 3.104.

Figura 3.104 Pestaña "Propiedades" seleccionada

En el campo "Patrón" escribimos "END OF DATA" que es el mensaje que queremos diferenciar. En el campo "Invertir coincidencia" seleccionamos "True". De esta forma se reconocerá esta descripción para esta pantalla cuando en el campo "INQC-MSG" aparezca cualquier cosa excepto el mensaje indicado.

Ahora tenemos que crear una descripción parea esta pantalla que reconozca cuando aparece el mensaje "END OF DATA". Para ello hacemos clic derecho sobre la "Descripción" y seleccionamos "Añadir nueva descripción" como se puede ver en la Figura 3.105.

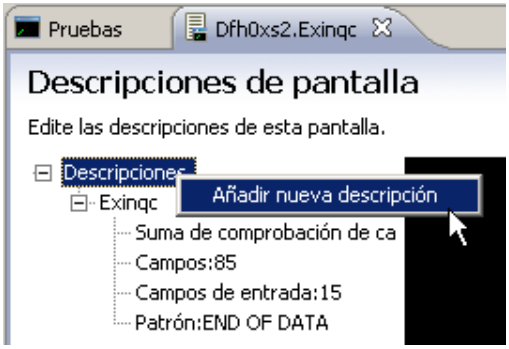


Figura 3.105 Añadir nueva descripción

Nos aparecerá la ventana de la Figura 3.106 donde en el campo "Identificar como:" le damos un nombre a la descripción, por ejemplo HasMsg, y en el campo "Copiar de:" seleccionamos "Exinqc" para que haga una copia de esta descripción.

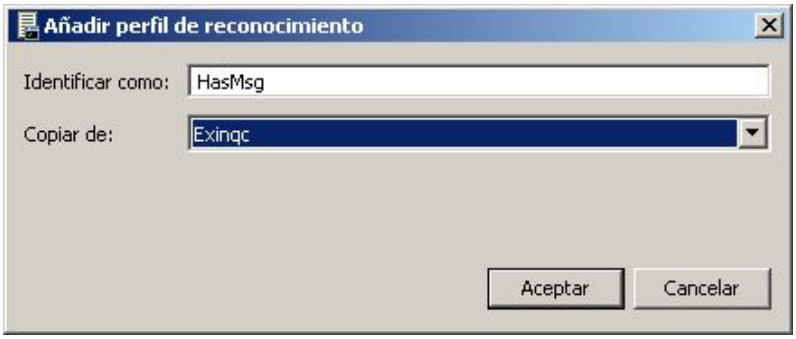


Figura 3.106 Características de la copia de la descripción

Nos aparecerá una nueva descripción con el nombre HasMsg. Seleccionamos el "Patrón" de este campo y en las "Propiedades" que hemos visto en la Figura 3.104 podemos el campo "Invertir coincidencia" a "False". De esta forma se reconocerá esta descripción de esta pantalla cuando aparezca en el campo "INQC-MSG" el mensaje "END OF DATA". Guardamos los cambios realizados con el botón de la Figura 3.60. Si ahora navegamos por estas pantallas veremos como reconoce distinta descripción si tenemos el mensaje "END OF DATA" o si no lo tenemos como podemos ver en la parte superior derecha de las Figuras 3.107 y 3.108 respectivamente.

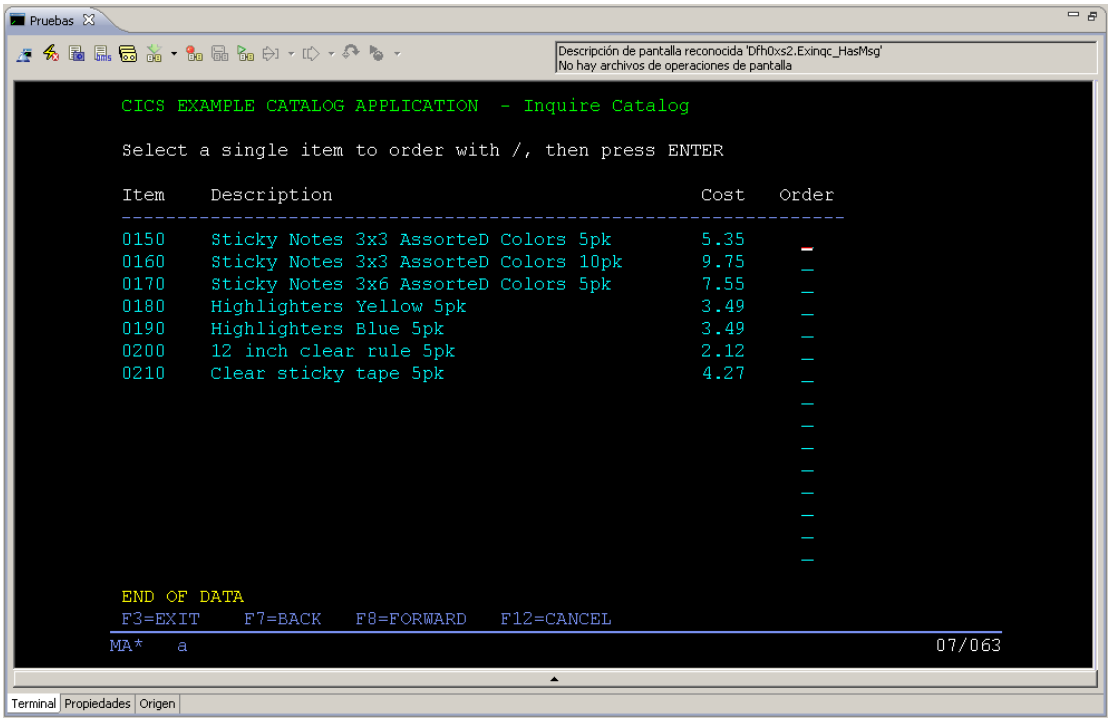


Figura 3.107 Pantalla con mensaje END OF DATA

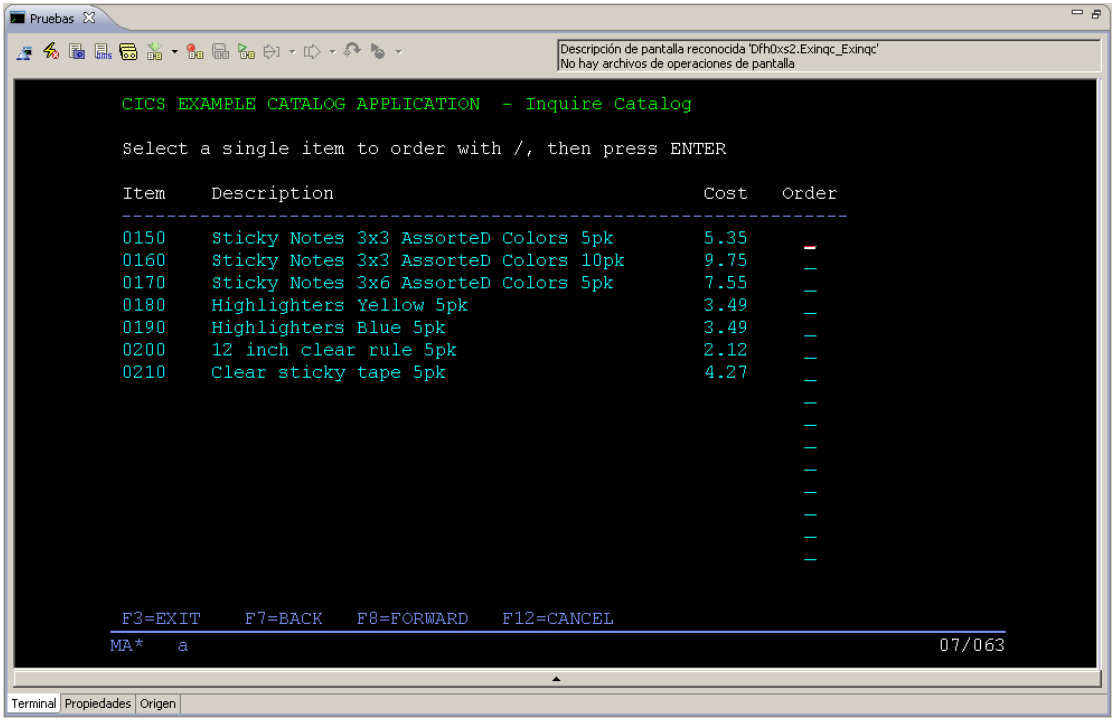


Figura 3.108 Pantalla sin mensaje END OF DATA

Una vez que tenemos las pantallas preparadas podemos empezar a grabar el flujo con el bucle.

Empezamos a grabar el flujo como hemos visto anteriormente aunque ahora no crearemos variables de entrada sino que directamente escribiremos un 1 en el campo "Action" y pulsamos Control. De esta forma llegamos a la primera pantalla del listado que podemos ver en la Figura 3.93. Una vez en esta pantalla pulsamos el botón de la Figura 3.109 para empezar a grabar un flujo.

Figura 3.109 Botón "Iniciar bucle"

Nos aparecerán las ventanas para crear una nueva variable que controlará el bucle. Creamos una variable por ejemplo con el nombre "var_Bucle". Ahora ya podemos empezar a grabar las acciones que se realizarán en cada iteración del bucle. Extraemos los datos de la pantalla en una variable de salida por ejemplo de nombre "listado". Vemos en la Figura 3.110 como la pantalla de selección de la variable de salida ha cambiado ligeramente. Nos aparece un campo para que indiquemos el "Número máximo de iteraciones". Esto se hace para que, en caso de producirse alguna situación que cree un bucle infinito, al llegar al número que le indiquemos el bucle finalizará. Le damos el valor 20 que es más que suficiente para el listado que queremos obtener.

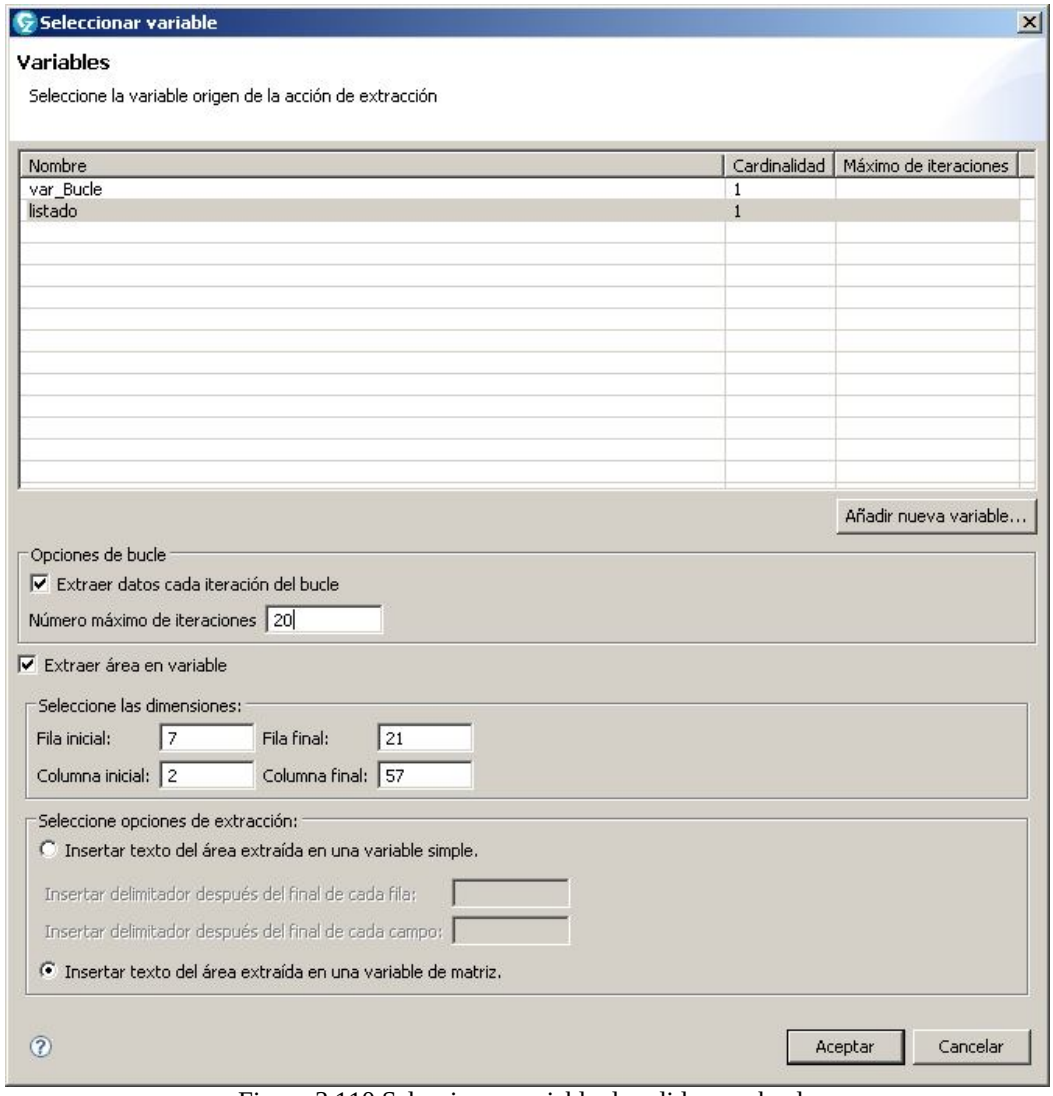


Figura 3.110 Seleccionar variable de salida para bucle

Una vez que hemos creado y seleccionado la variable de salida pulsamos F8 para pasar a la siguiente pantalla del listado. En la pantalla del listado que nos aparece pulsamos el botón de la Figura 3.111 para indicar que estas acciones, extraer los datos de la pantalla y pulsar F8, son las que forman una iteración del bucle.

Figura 3.111 Fin de iteración de bucle

La pantalla donde finaliza una iteración tiene que ser la misma que donde empieza. En caso de no ser así nos aparecerá la ventana de la Figura 3.112 indicándonos este hecho y no nos dejará indicar el fin de a iteración.

Figura 3.112 Error al finalizar la iteración del bucle

Seguimos pulsando F8 hasta llegar a la pantalla de fin de datos del bucle, que en este caso llegaremos tras pulsar una vez. Una vez en ella pulsamos el botón de la Figura 3.113 que indica el final del bucle.

Figura 3.113 Botón para finalizar bucle

Nos aparecerá la ventana de la Figura 3.114 donde podemos elegir la forma del bucle. Normalmente tendremos que elegir la que acaba en Exinqc, sin InitialExtract. En la parte inferior vemos como es la forma del bucle. Si estamos en una pantalla con la descripción Exinqc y tras una iteración llegamos a otra pantalla con la misma descripción continuamos el bucle. Si llegamos a una pantalla con la descripción HasMsg finalizamos el bucle que es lo que queremos.

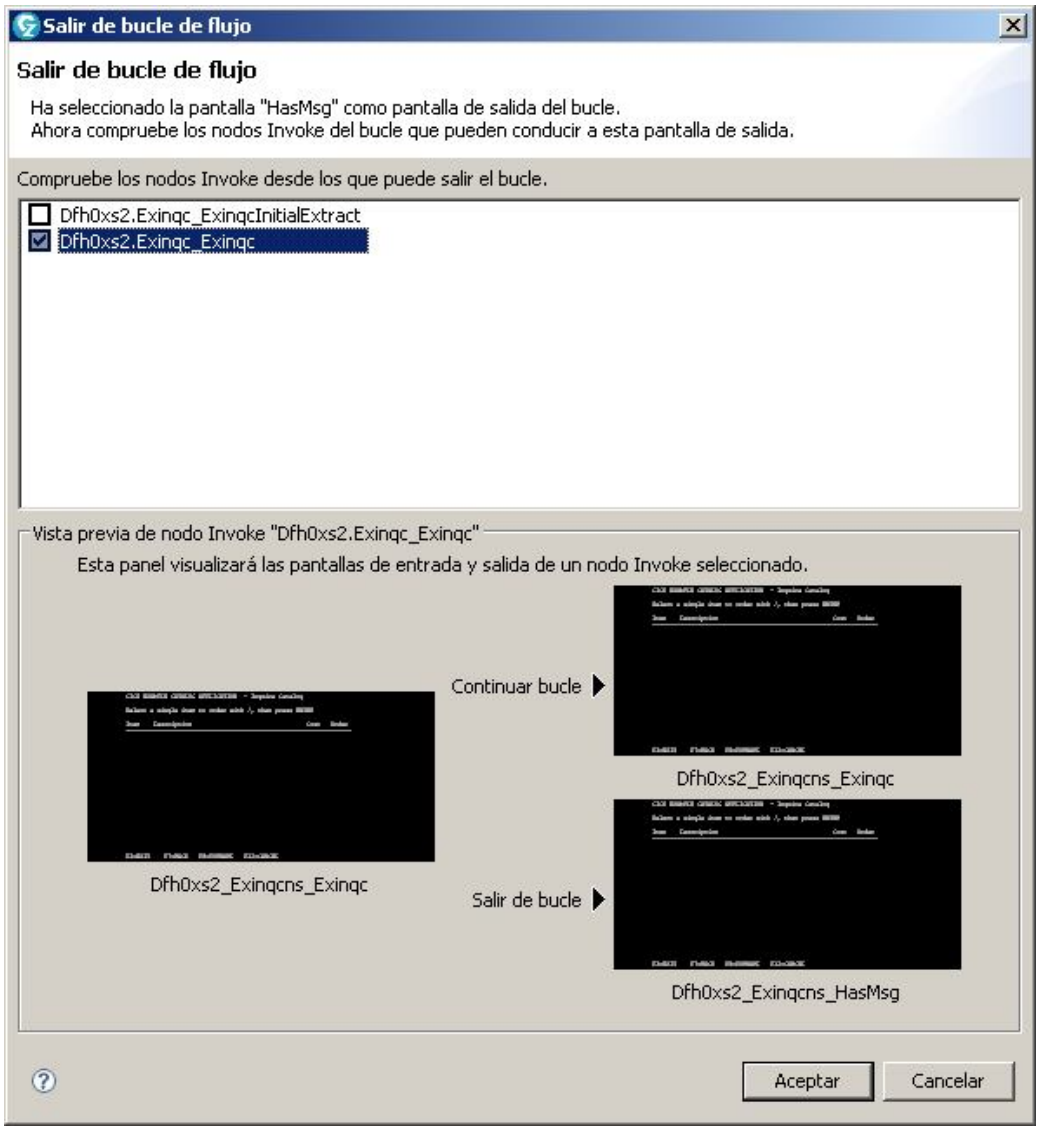


Figura 3.113 Ventana de selección de la forma del bucle

Una vez hecho esto ya podemos finalizar la grabación del bucle y guardarlo.

Si una vez instalado en el CICS lo probamos, ya sea por medio del CTG, como de un servicio Web llamado desde una aplicación de escritorio o desde el RDz, obtenemos todos los datos del listado y no únicamente los de la primera pantalla del listado. De esta forma comprobamos que hemos grabado el bucle correctamente.

3.6.1.- Grabación de bucle con una pantalla de listado

En algunos casos puede que nos encontremos con que queremos grabar un bucle para un listado pero no tenemos más que una pantalla de datos. En un principio esto puede ser un problema ya que al pulsar F8 al empezar a grabar el bucle, no llevaría a la pantalla de final de datos por lo que no le podríamos decir cuando finaliza una iteración. Para solucionar esto, en la primera pantalla del listado, una vez que hemos empezado a grabar el bucle, pulsamos F7. Esto nos llevaría a la pantalla anterior del listado pero al estar en la primera normalmente nos dejará en la misma página. Pulsamos el botón de la Figura 3.111 para finalizar la iteración y navegamos hasta la página de fin de datos y finalizamos la grabación del bucle de la manera normal.

Una vez hecho esto, vamos a cambiar la tecla de cambio de pantalla de la F7 por la F8. Para ello, hacemos doble clic en el flujo para que se abra en edición con lo que nos aparecerá la pantalla de la Figura 3.114.

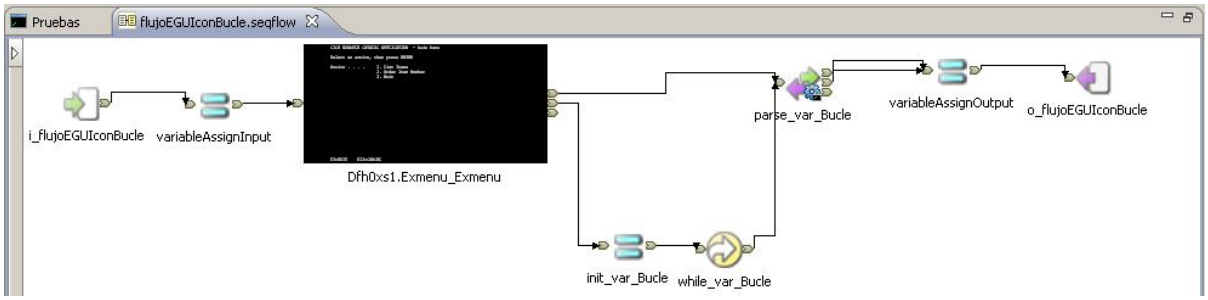


Figura 3.114 Flujo con bucle

El nodo "while_var_Bucle" representa el bucle y si hacemos doble clic en él se abrirá el bucle grabado en edición como podemos ver en la Figura 3.115.

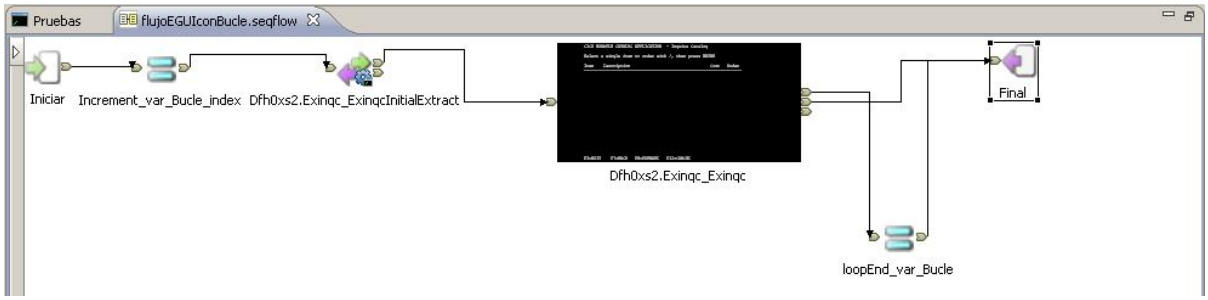


Figura 3.115 Bucle del flujo anterior

Para cambiar la tecla F7 por F8 hacemos clic derecho sobre el nodo que representa la pantalla del listado y elegimos "Abrir rutina de correlación → Dfh0xs2.Exinqc_Exinqc" como podemos ver en la Figura 3.116.

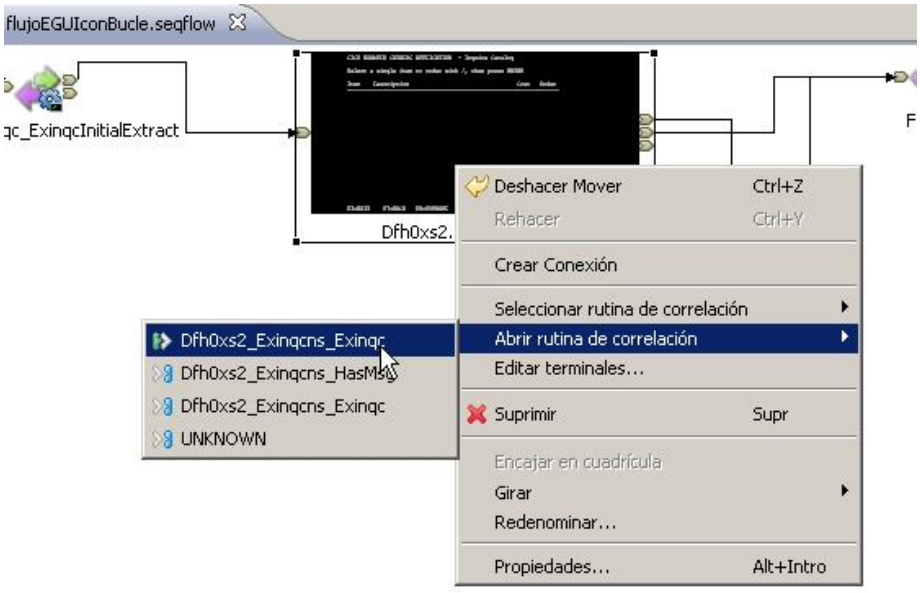


Figura 3.116 Abrir rutina de correlación de la pantalla del listado

De esta forma se abrirá la ventana de la Figura 3.117 donde podemos ver todas las asignaciones que se hacen sobre la pantalla del listado. Si tuviésemos asignaciones de variables a campos,

como en la pantalla inicial del bucle básico, o asignaciones de valores directamente a campos, como en el campo "Action" de la pantalla inicial de este flujo, lo veríamos reflejado aquí y podríamos hacer modificaciones.

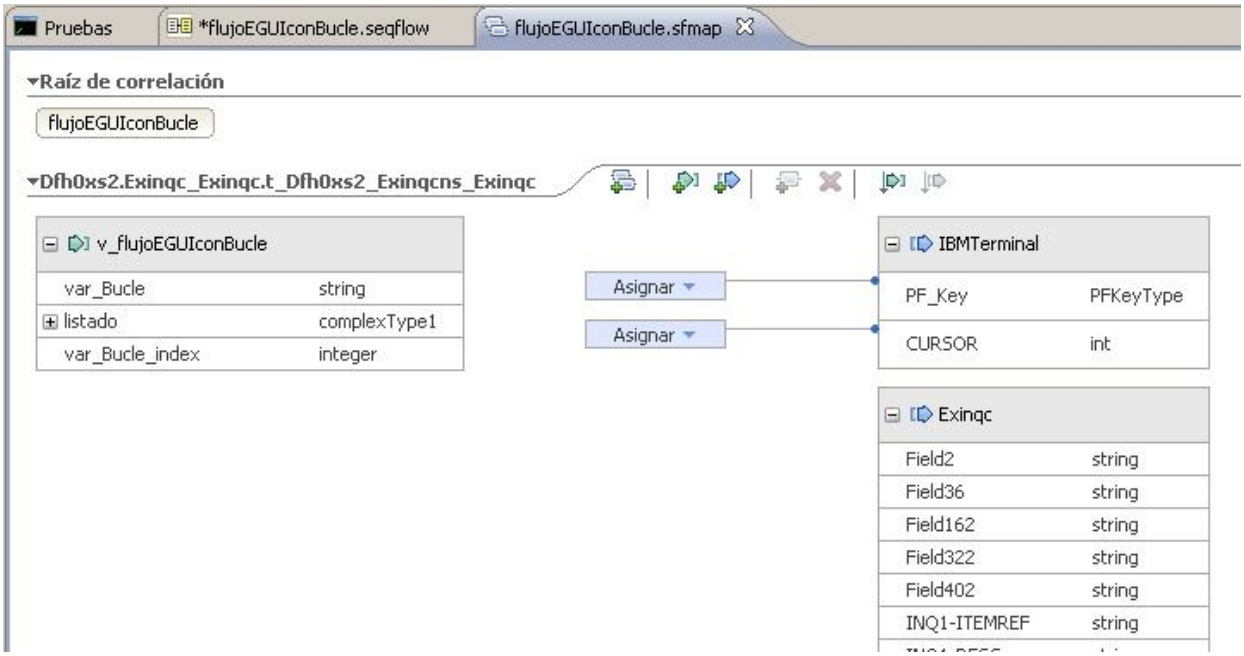


Figura 3.117 Rutina de correlación de la pantalla de listado

Seleccionamos el recuadro "Asignar" que está unido al campo PF_Key del grupo IBMTerminal. Este campo es el que nos dice que tecla PF, como las F1-F2, tecla Intro, etc., se va a pulsar cuando nos encontremos en esta pantalla. Una vez seleccionado, en la parte inferior de la pantalla de RDz seleccionamos la pestaña "Propiedades" y dentro de esta la pestaña "General". En la Figura 3.118 podemos ver como el valor que se está asignando al campo PF_Key es PF7 o F7.

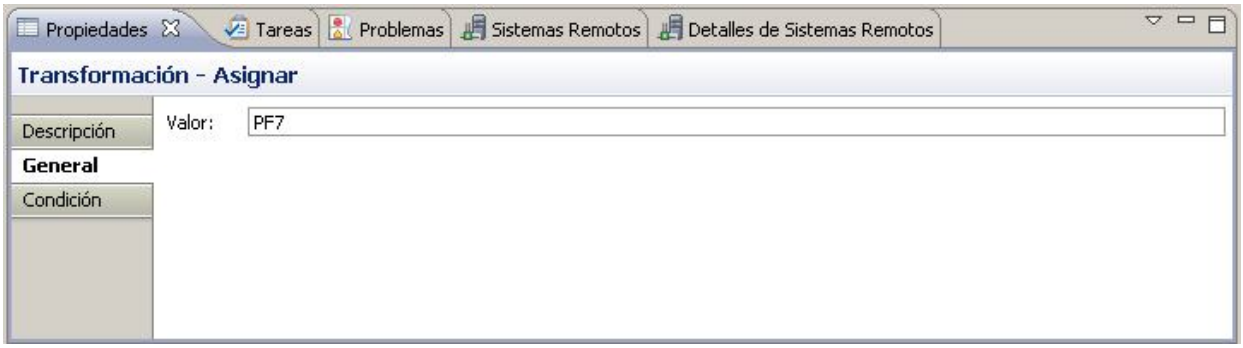


Figura 3.118 Asignación para el campo PF_Key

Cambiamos este valor por PF8 para que el listado en vez de retroceder, lo que hace con la tecla F7, avance que es lo que hace con la tecla F8. Una vez realizado este cambio ya podemos guardar los cambios realizados con el botón de la Figura 3.60.

Si instalamos el flujo y lo probamos podemos ver como obtenemos todos los datos del listado al igual que con el flujo anterior, por lo que los cambios que hemos realizado funcionan para poder realizar un bucle de un listado a partir de una única pantalla de datos.

3.7.- Elegir elemento de una página concreta

En algunos casos puede que nos encontremos con que queremos obtener más datos de un elemento de un listado. Si los diferentes elementos del listado están numerados consecutivamente no habría ningún problema porque con introducir el número del listado en el campo correspondiente en cualquier pantalla del listado tendríamos el flujo listo. El problema viene cuando algunos programas CICS numeran los elementos de un litado del 1 al N pero en la siguiente página del listado vuelven a numerarlos del 1 al N, en vez de empezar por el N+1. En estos casos no es lo mismo introducir el elemento 8 en la pantalla 1 que en la 2 o la 3. Para ello, tenemos que hacer que el flujo finalice la ejecución del bucle en la pantalla que nosotros queramos, y en esa pantalla introducir el número del elemento deseado.

Vamos a ver un ejemplo de cómo realizar esto. Para ello vamos a utilizar la transacción FRA0 que es una transacción de desarrollo propio, no viene suministrada con el CICS ni es una transacción de ejemplo. Esta transacción permite, entre otras cosas, visualizar las declaraciones de un determinado DNI para un determinado año y, una vez listadas, seleccionar una de ellas para ver más datos. Esta transacción tiene seguridad por lo que antes de empezar a utilizarla tenemos que identificarnos en el CICS con la transacción CESN o de lo contrario nos aparecerá el mensaje de la Figura 3.119 al intentar ejecutar la transacción.

```
ANTES DE ACCEDER A LA TRANSACCION FRA0 DEBERA HACER CESN
```

Figura 3.119 Error al intentar ejecutar la FRA0 sin identificarnos previamente

Importamos el grupo de mapas MFFQ0P, como hemos visto anteriormente, que contiene los mapas que vamos a utilizar en esta transacción. Las pantallas correspondientes a los mapar que vamos a utilizar son los siguientes:

- Pantalla inicial de la transacción FRA0 en la Figura 3.120. Mapa Mffq0p.Mfra2p.

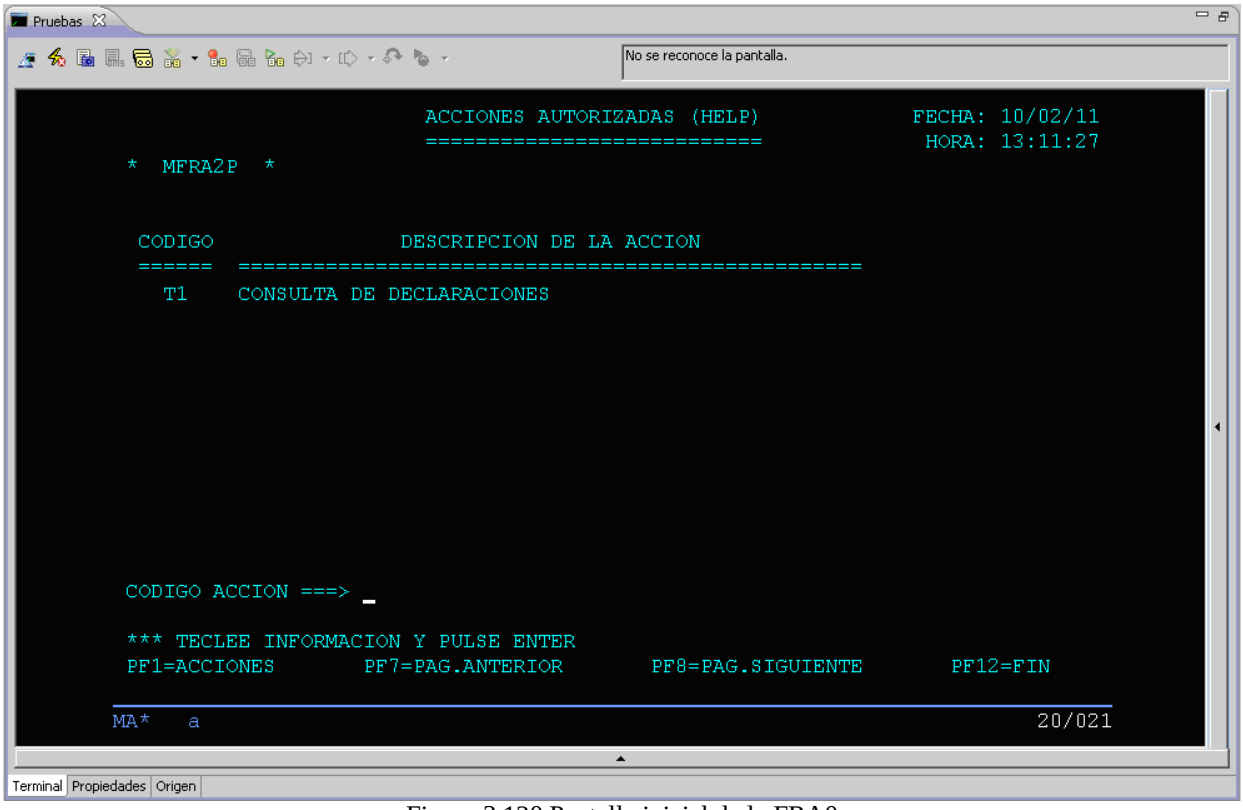


Figura 3.120 Pantalla inicial de la FRA0

- Pantalla inicial de la opción T1 en la Figura 3.121. Mapa Mffq0p.Mffq1p.

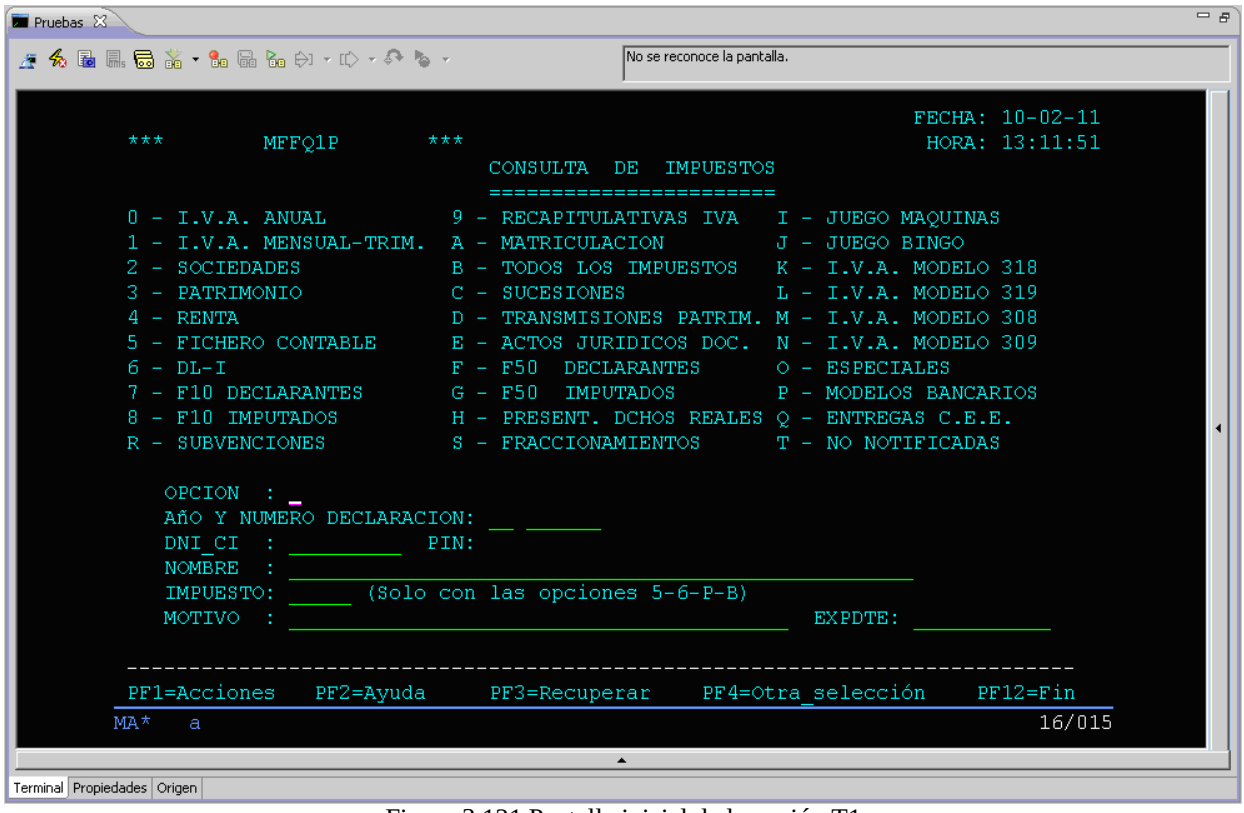


Figura 3.121 Pantalla inicial de la opción T1

- Pantalla de avisos de notificaciones en la Figura 3.122. Mapa Mffq0p.Mffq1r.

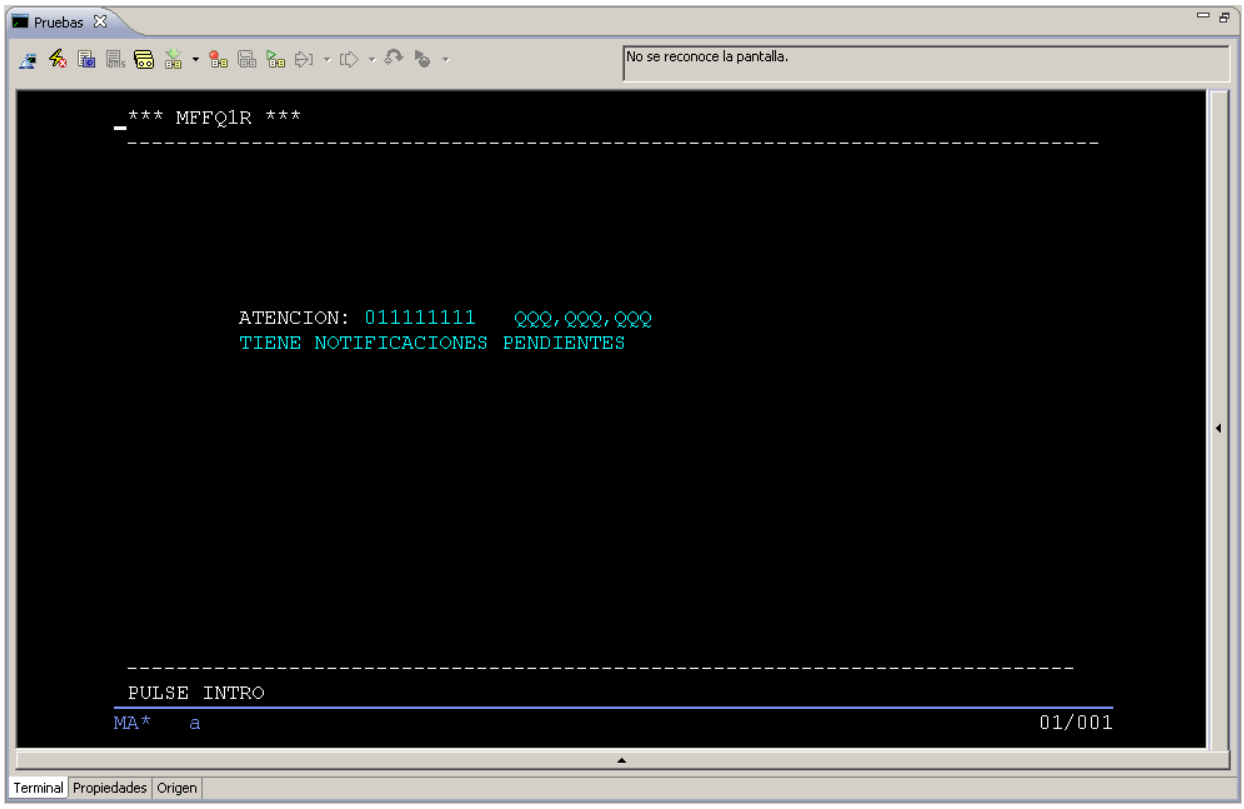


Figura 3.122 Pantalla de avisos de notificaciones

- Pantalla de listado de declaraciones en la Figura 3.123. Mapa Mffq0p.Mffq2p.

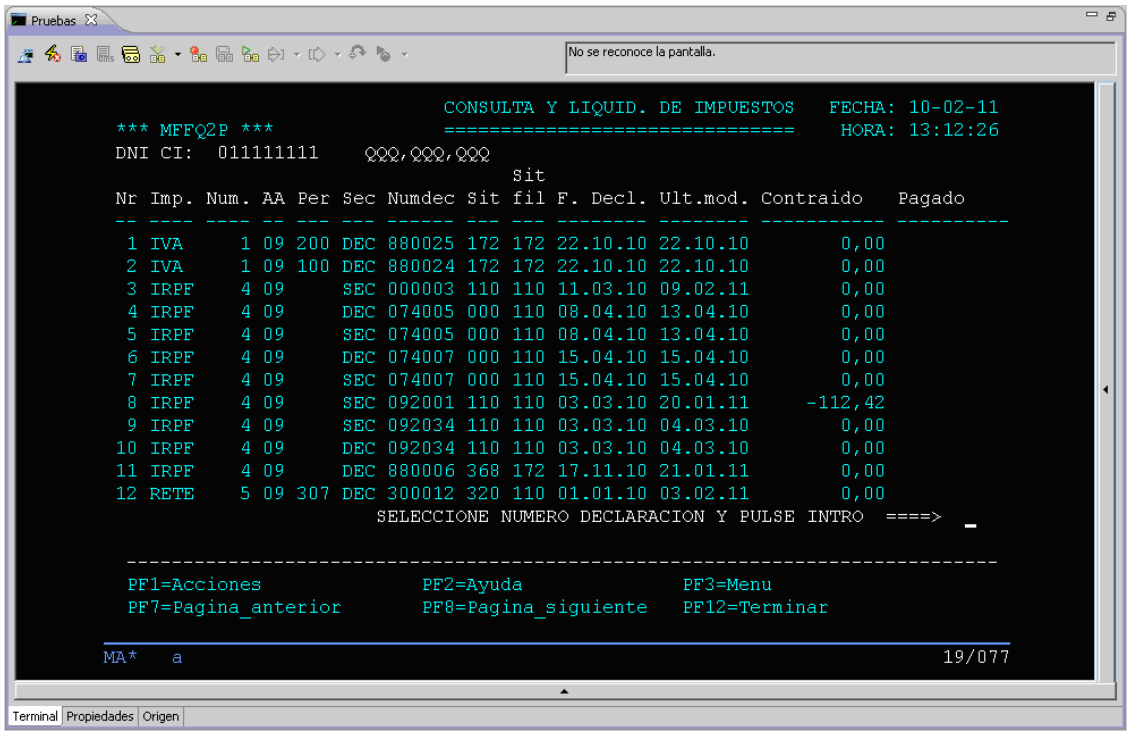


Figura 3.123 Pantalla de listado de declaraciones

- Pantalla de detalles de una declaración en la Figura 3.124. Esta pantalla no la reconoce por lo que la capturamos con el método que hemos visto anteriormente y le damos el nombre “DETALLES”

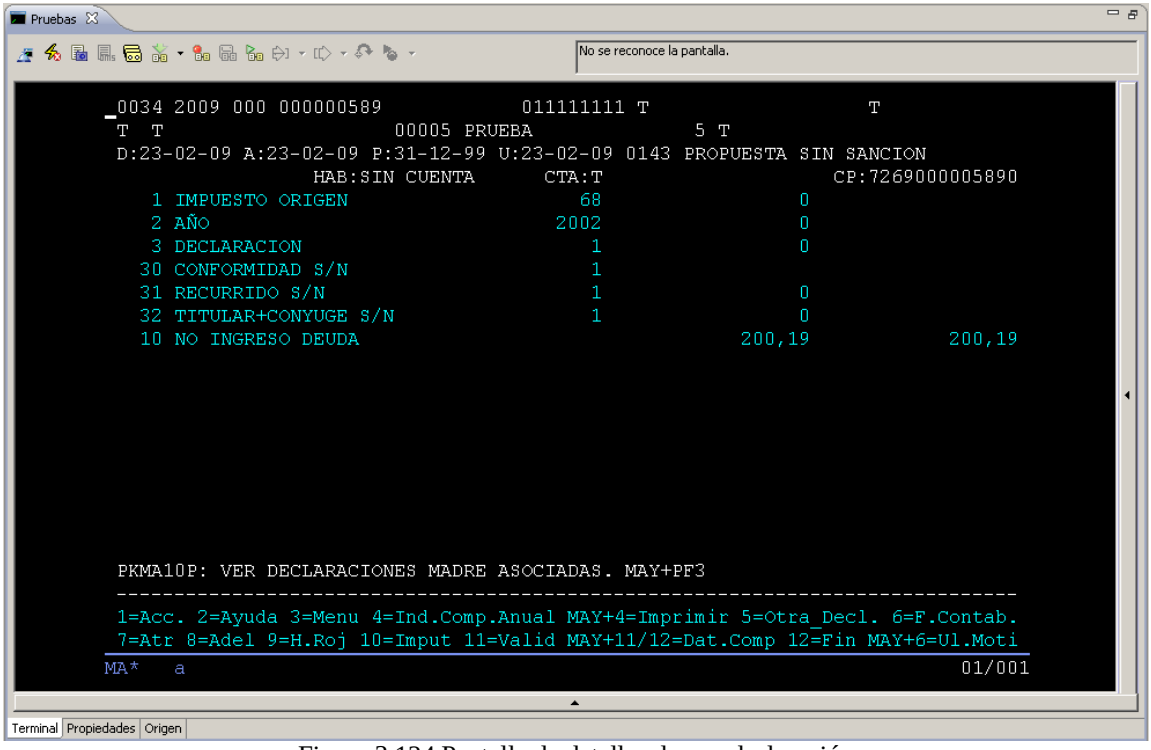


Figura 3.124 Pantalla de detalles de una declaración

Recorremos todas estas pantallas capturándolas como hemos visto anteriormente. A cada una le damos el nombre que aparece en la parte superior izquierda que es el nombre de la pantalla. A la última, que no tiene nombre, le damos el nombre "DETALLES".

Una vez que tenemos capturadas las pantallas, tenemos que preparar la pantalla de la Figura 3.123 para que reconozca el fin del listado como hemos visto anteriormente. El mensaje de final de listado es "FIN DE DATOS" como podemos ver en la Figura 3.125.

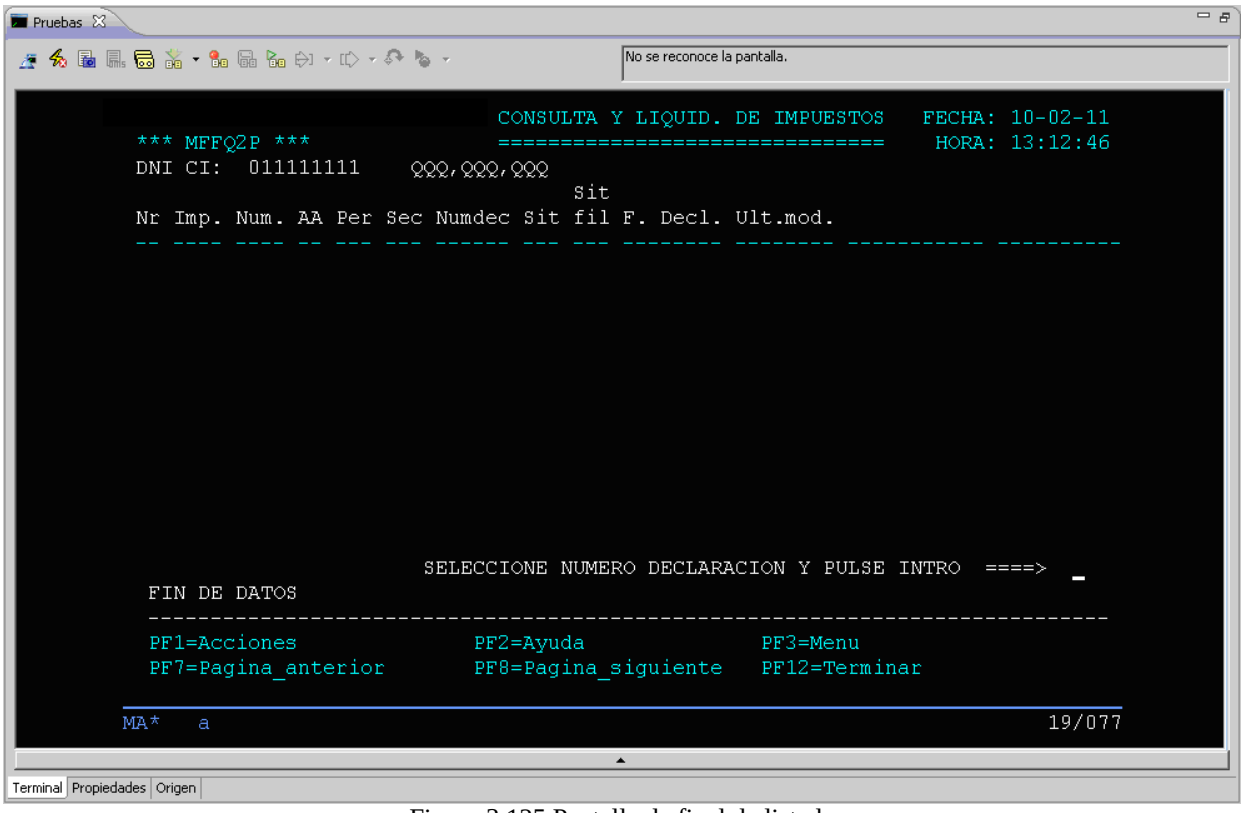


Figura 3.125 Pantalla de final de listado

Una vez que tenemos preparadas todas las pantallas, podemos empezar a grabar el flujo de nombre "flujoElementoDePag". Empezamos a grabar el flujo y en la pantalla de la Figura 3.120 introducimos de forma estática, sin crear una variable de entrada, el "código de acción" "T1" con lo que nos llevará a la pantalla de la Figura 3.121. En esta pantalla creamos dos variables de entrada para los campos "dni_ci" e "IMPUESTO" que se llamen respectivamente "dni" e "impuesto". Una vez creadas las variables de entrada introducimos los datos "B" en "OPCION", de forma estática, y "011111111" y "4" en "DNI_CI" e "IMPUESTO" respectivamente. Pulsamos Control lo que nos llevará a la pantalla de la Figura 3.122. En esta pantalla no tenemos que realizar ninguna acción y pulsamos Control para llegar a la pantalla de la Figura 3.123 donde creamos un bucle para extraer todos los datos del listado como hemos visto anteriormente. Le decimos que el número máximo de iteraciones sea de 20. Una vez que hemos hecho esto finalizamos la grabación del flujo y lo guardamos.

Una vez grabado el flujo, vamos a crear una variable para indicar al flujo en que pantalla se tiene que parar del listado. Esta pantalla será la siguiente a la que pantalla de la que queramos pedir los detalles de un elemento. Primero tenemos que crear una variable de entrada. El grupo de variables de entrada para este flujo se llama "i_flujoEleemntodDePag.sfmxsd" como podemos ver señalado en la Figura 3.126 y se encuentra en "pruebasEGUI.Interface → Mensajes". Además podemos ver todos los grupos de variables que tenemos hasta el momento.

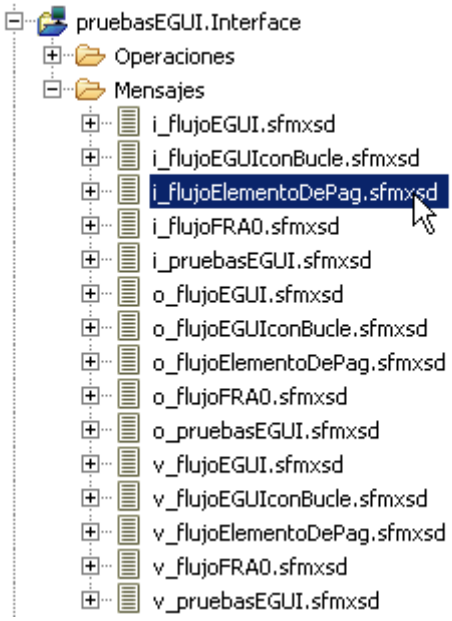


Figura 3.126 Grupos de variables de todos los flujos

Hacemos doble clic sobre este grupo de variables y, una vez que se habrá para que lo podamos editar, hacemos clic derecho sobre el nodo con nombre "i_flujoElementoDePag" y elegimos "Añadir elemento" como podemos ver en la Figura 3.127 para añadir una nueva variable a este grupo.

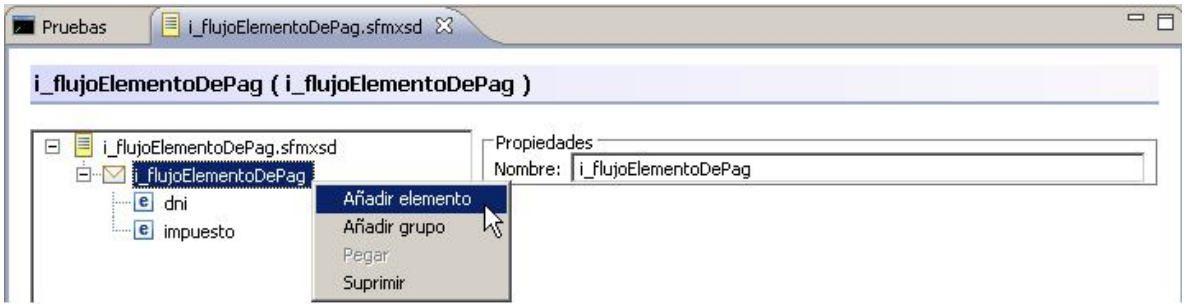


Figura 3.127 Añadir nueva variable

Esta nueva variable le damos el nombre "numPagina" y le decimos en las propiedades que sea de tipo "integer" y de 2 dígitos, lo que nos va a permitir llegar hasta la página 99 del listado. Esto lo podemos ver en la Figura 3.128.

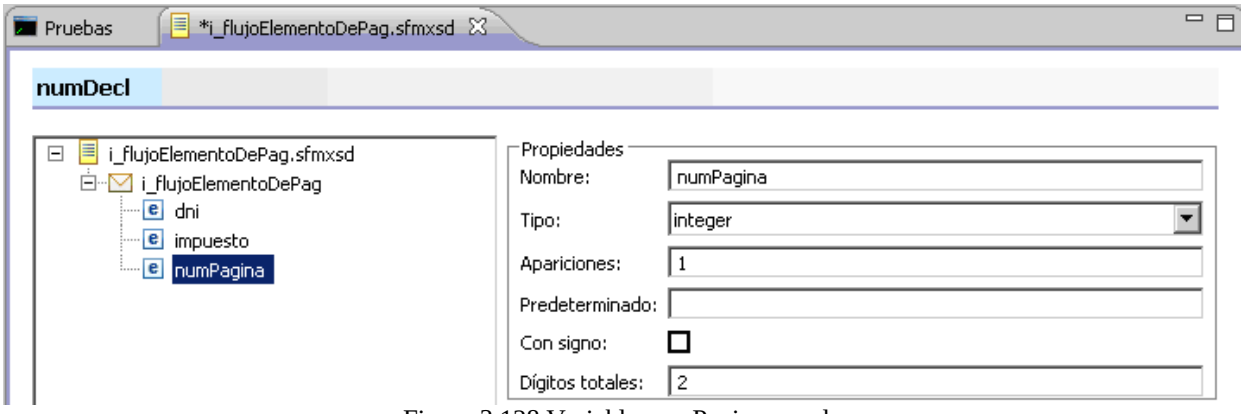


Figura 3.128 Variable numPagina creada

Realizamos los mismos pasos sobre el grupo de variables "v_flujoElementoDePag" que es el grupo de variables auxiliares del flujo. Le damos a la variable el mismo nombre "numPagina". Guardamos los cambios que hemos realizado como por medio del botón de la Figura 3.65.

Una vez hecho esto, tenemos que asignar la variable de entrada "numPagina" a la variable auxiliar con el mismo nombre. Para ello, hacemos doble clic sobre el flujo para abrirlo en edición. Veremos un nodo llamado variableAssignInput que es donde se asignan las variables de entrada a las variables auxiliares como se puede ver en la Figura 3.129.

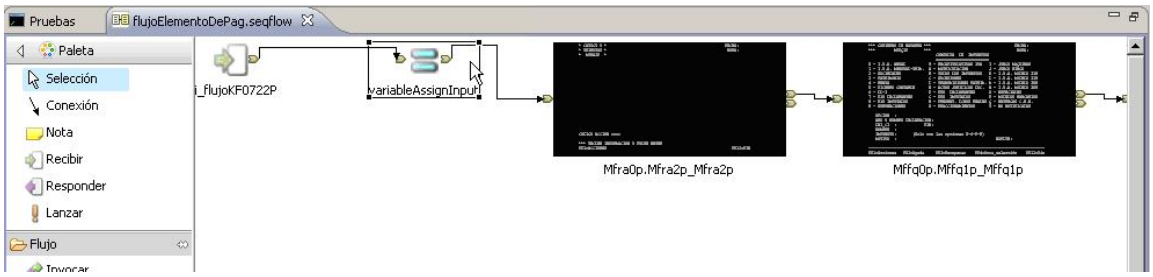


Figura 3.129 Nodo variableAssignInput

Nos aparecerá la pantalla de la Figura 3.130 donde podemos ver las asignaciones que hay entre las variables de entrada y las variables auxiliares.

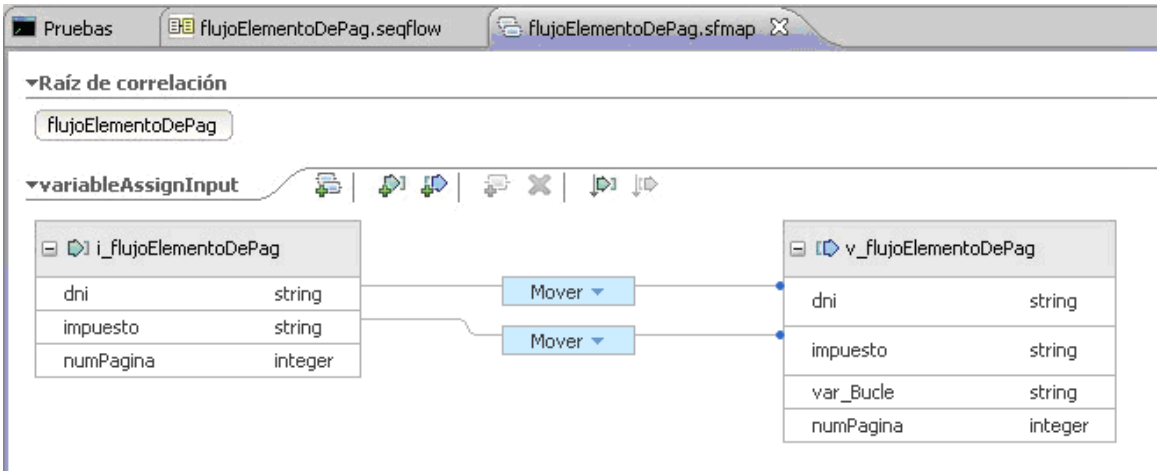


Figura 3.130 Asignación de las variables de entrada a las variables auxiliares

Para asignar una variable a otra pinchamos la variable de entrada numPagina y arrastramos la línea que nos aparecerá sobre la otra variable de forma que queden conectadas como vemos en la Figura 3.131.

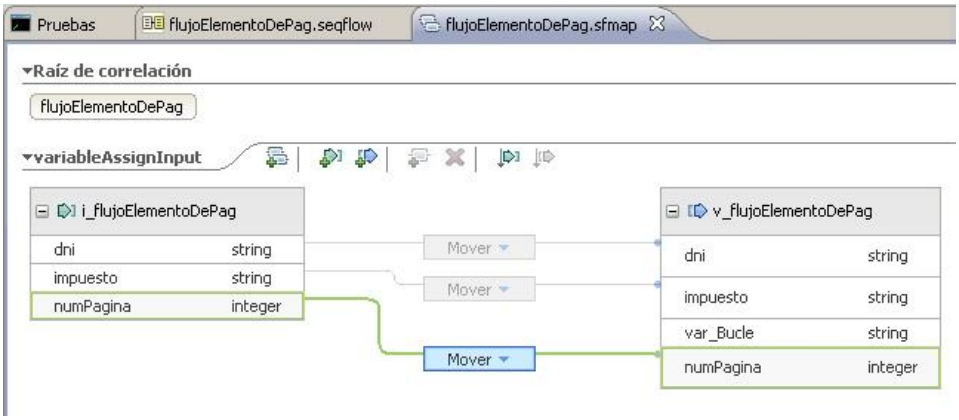


Figura 3.131 Asignación de la variable de entrada creada a la variable auxiliar

Ahora vamos a hacer que el flujo se detenga en la página del listado que deseemos. Para ello tendremos que editar la expresión que hace que el bucle finalice sin haber llegado a la pantalla de fin de bucle. Hacemos clic derecho sobre el nodo whiel_var_Bucle y seleccionamos "Editar expresión" como vemos en la Figura 3.132.

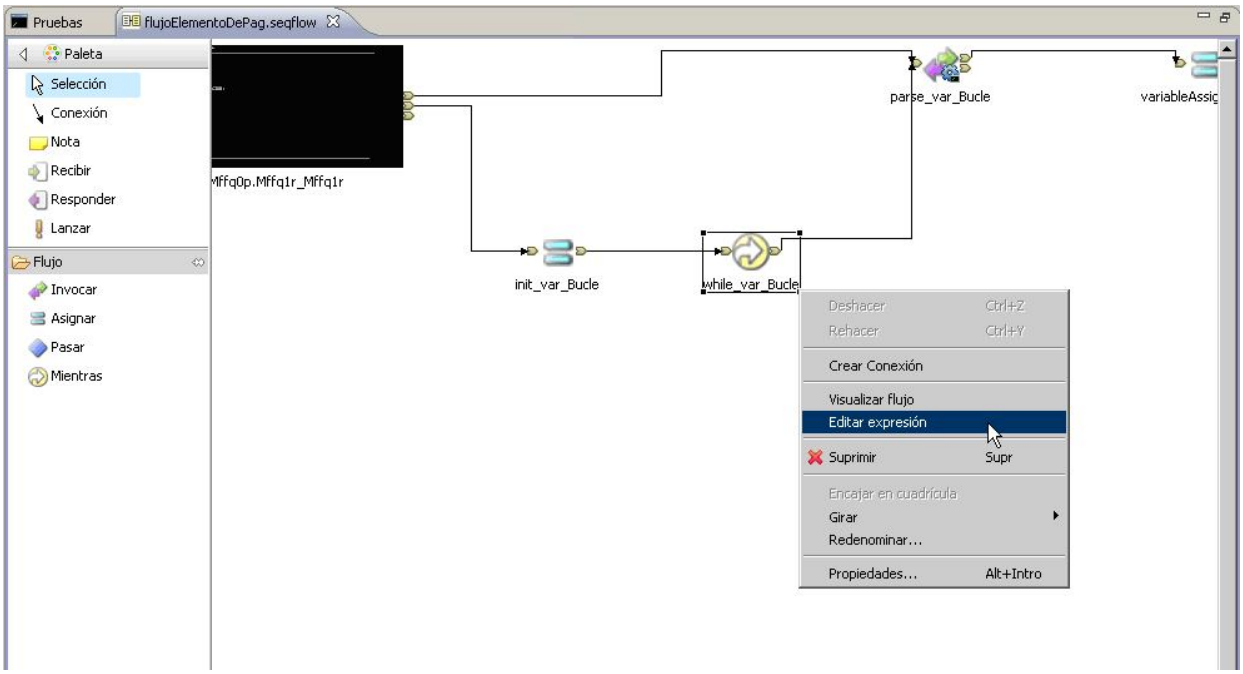


Figura 3.132 Editar expresión de final de bucle

Nos aparece la ventana de la Figura 3.133 donde vemos que el bucle continuara haciendo iteraciones mientras var_Bucle sea TRUE o mientras el índice sea menor que el número de iteraciones máximo que le hemos dicho en la variable que hemos definido para extraer los datos. Se pone a FALSE dentro del bucle cuando detecta que ha llegado a la pantalla que le hayamos dicho que es la de final del bucle o cuando haya llegado al número máximo de iteraciones que en este caso es 20.

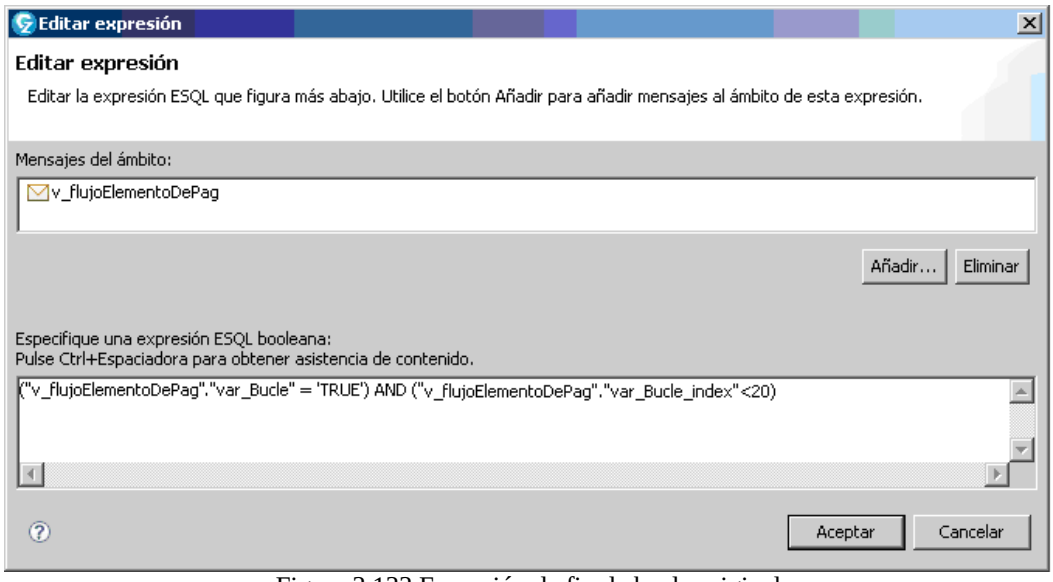


Figura 3.133 Expresión de fin de bucle original

Modificamos esta expresión para hacer pare cuando llegue a la iteración, y por extensión al número de pantalla, que le hayamos pasado en la correspondiente variable de entrada. Cabe destacar que se detiene cuando alcanza la pantalla siguiente al número que le digamos ya que el contador empieza en 0 pero en la iteración cero extrae los datos de la pantalla 1 por lo que tenemos que pasarle un número que sea una unidad menor a aquella pantalla en la que nos queremos quedar, restarle uno al número que nos ha llegado o volver una pantalla atrás cuando acabe el bucle. Hemos optado por esta última opción ya que así esta pantalla ya la tenemos procesada y no tenemos que extraer sus datos antes de seguir con el flujo. La expresión quedaría de la manera que vemos en la Figura 3.134.

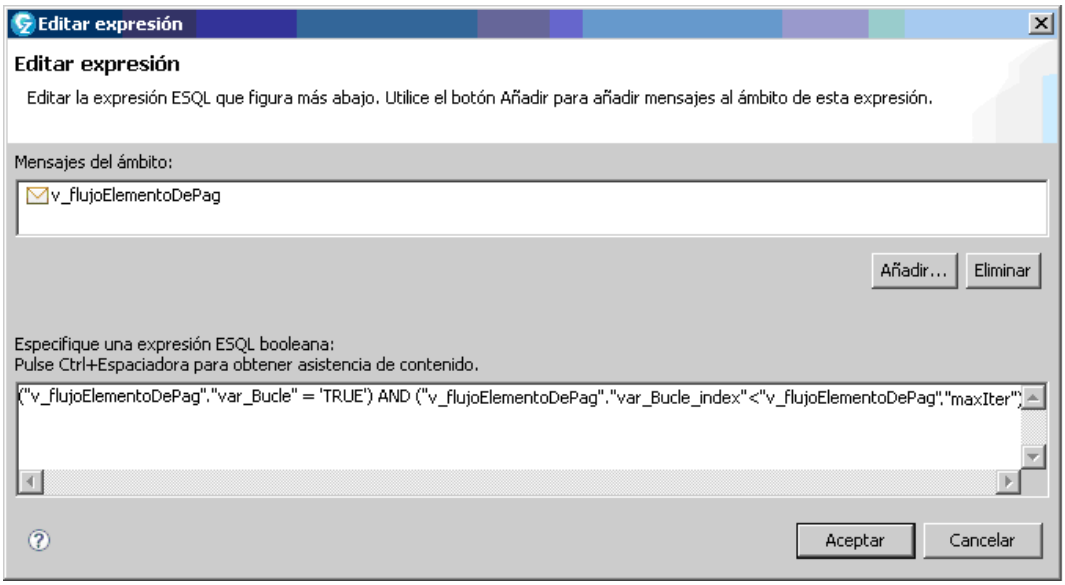


Figura 3.134 Expresión de final de bucle modificada

Guardamos los cambios como hemos dicho anteriormente.

Por último tenemos que volver una pantalla atrás de la que estamos, decirle de que elemento queremos obtener los detalles y extraer los datos que se nos muestren de ese elemento.

Para ello lo primero que vamos a hacer es situarnos en alguna de las pantallas del listado, nos da igual en cual mientras que no sea en la pantalla de fin de datos o fin de bucle, este caso lo trataremos luego, o la primera, ya que tenemos que volver una pantalla atrás. Por ejemplo nos situamos en la pantalla de la Figura 3.135 que es la segunda del listado.

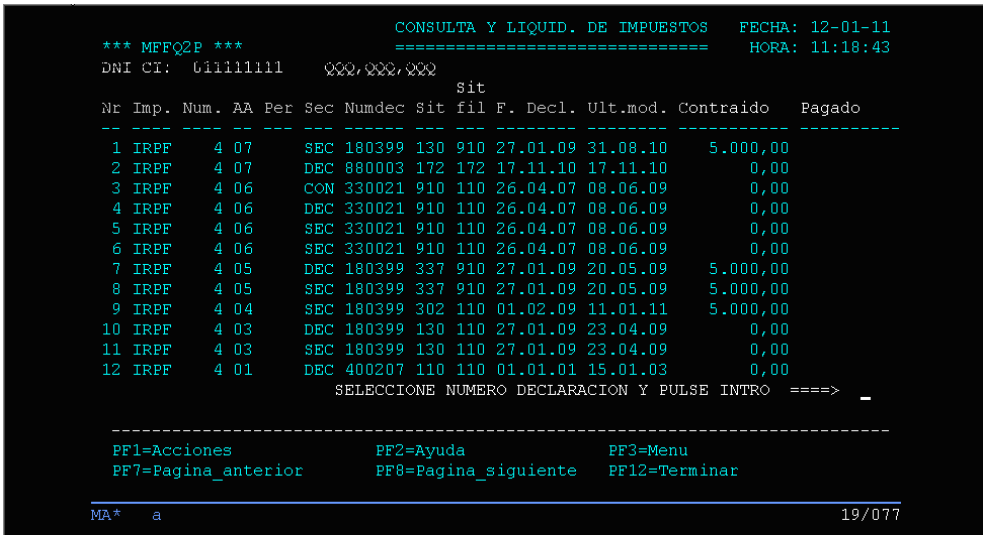


Figura 3.135 Segunda pantalla del listado

Cargamos el flujo que habíamos creado y empezamos la grabación del flujo. En este caso elegimos añadirlo al flujo cargado pero en una vía de acceso nuevo como podemos ver en la Figura 3.136. Esto hará que el nuevo flujo no se una a ningún nodo antes grabado.

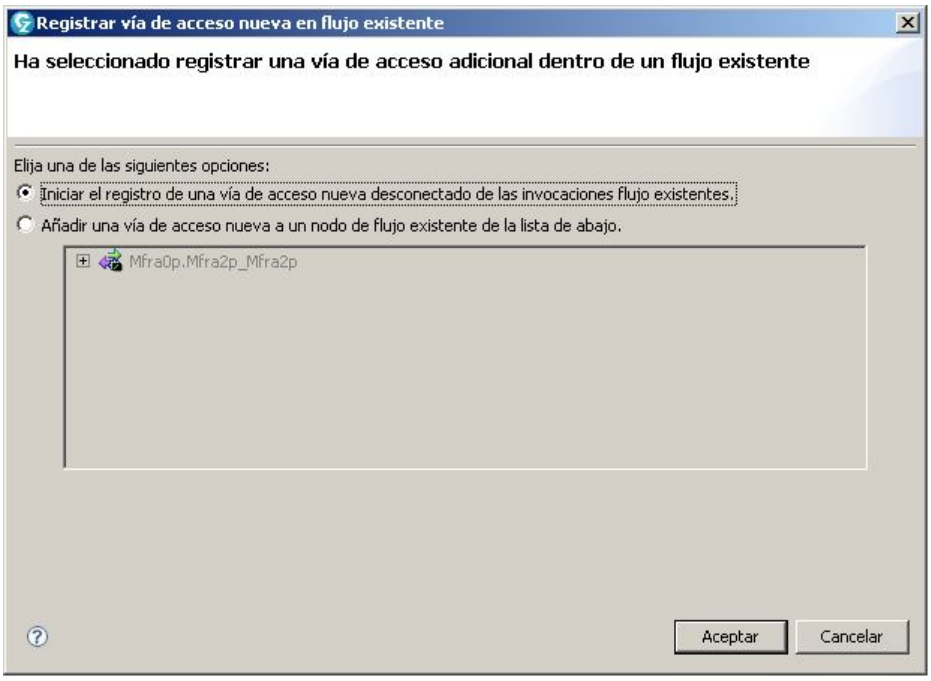


Figura 3.136 Grabar flujo en una vía de acceso nueva

Una vez hecho esto podemos grabar el flujo normalmente. El primer paso volver una pantalla atrás desde la que estamos lo que se hace pulsando F7.

A continuación creamos una variable de entrada para indicarle de que elemento queremos obtener los detalles. A esta variable le llamamos por ejemplo "numDecl" como podemos ver en la Figura 3.137.

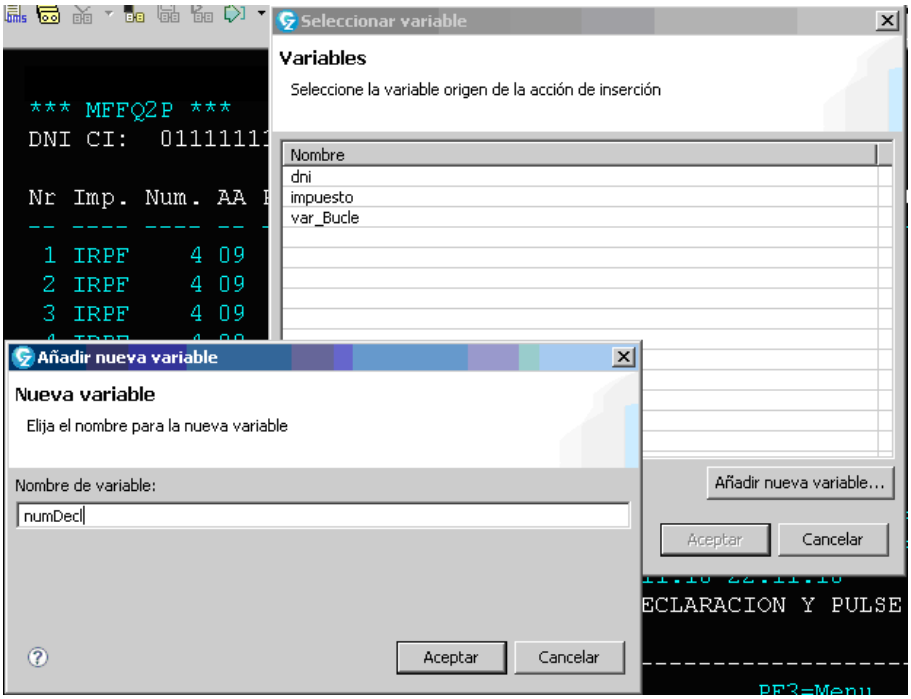


Figura 3.137 Variable numDecl para indicar el elemento del que queremos obtener detalles

Seleccionamos el campo de entrada al que vamos a vincular esta variable que es aquel donde nos dice “Seleccione numero de declaración y pulse intro”. La pantalla quedaría con el aspecto que podemos ver en la Figura 3.138 después de vincular la variable de entrada a su campo correspondiente.

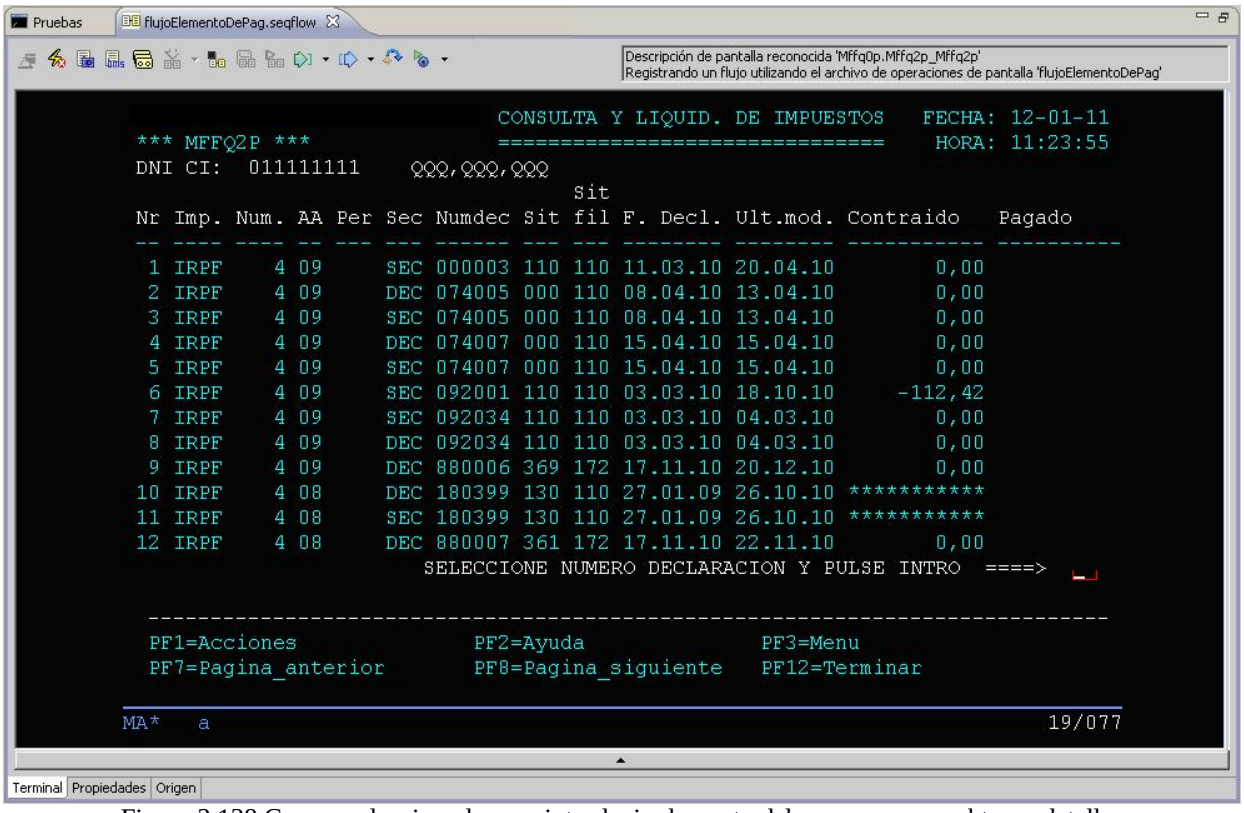


Figura 3.138 Campo seleccionado para introducir elemento del que queremos obtener detalles

Ahora introducimos un número de declaración en el campo correspondiente para poder seguir grabando el flujo, por ejemplo el 1, y pulsamos Control.

Nos aparecerá la pantalla de la Figura 3.124 de donde extraemos todos los datos en una variable de salida. Una vez hecho esto finalizamos la grabación del bucle y lo guardamos.

Lo siguiente que tenemos que hacer es unir este camino que acabamos de grabar de forma independiente con el flujo principal que habíamos grabado anteriormente. Veremos como nuestro flujo aparece con una cruz roja al lado como vemos en la Figura 3.129 debido a que tiene errores al tener nodos desconectados. Abrimos el editor de flujo con nuestro flujo como hemos dicho anteriormente.

Figura 3.139 Flujo con errores

Vemos como ha aparecido el camino que acabamos de grabar pero como está desconectado del resto del flujo como vemos en la Figura 3.140.

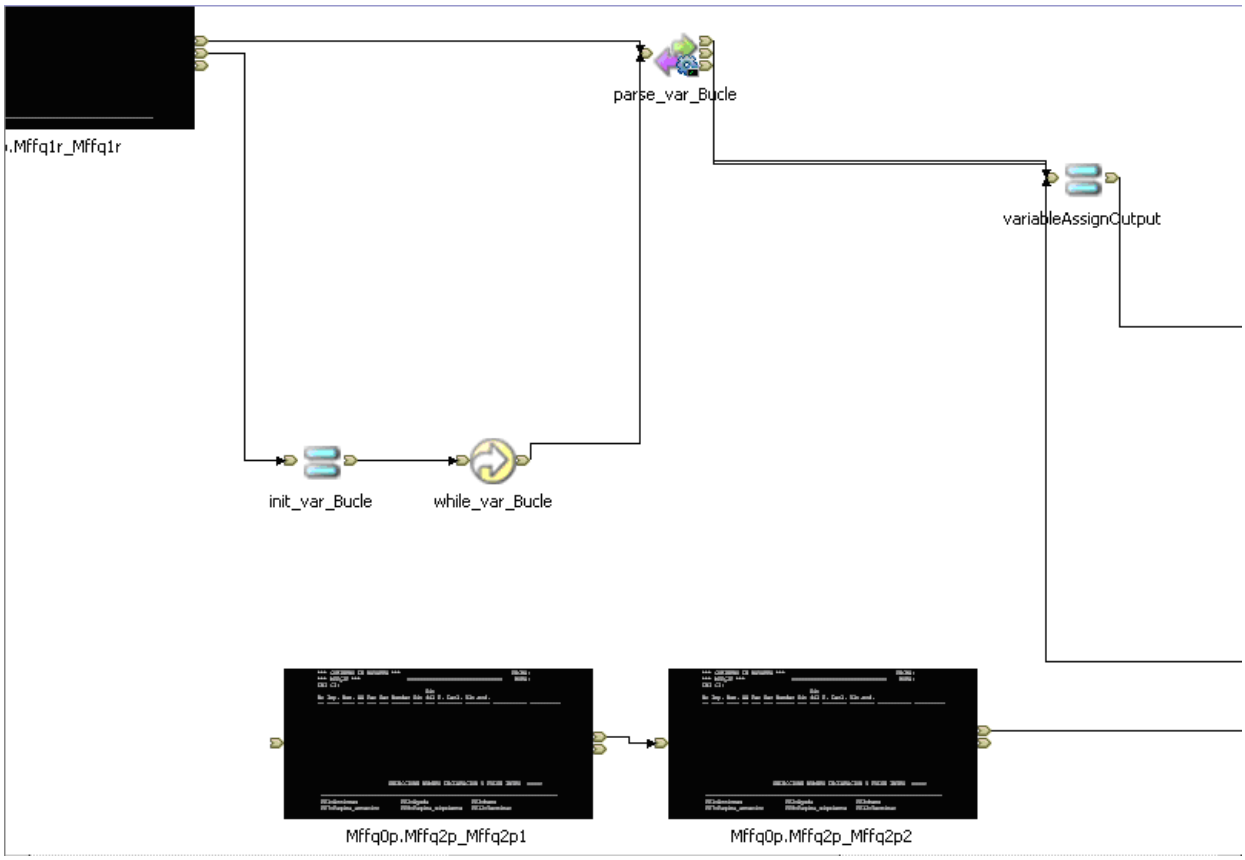


Figura 3.140 Nodos desconectados que acabamos de grabar

Para unir este camino con el flujo principal seleccionamos la conexión que podemos ver en la Figura 3.141.

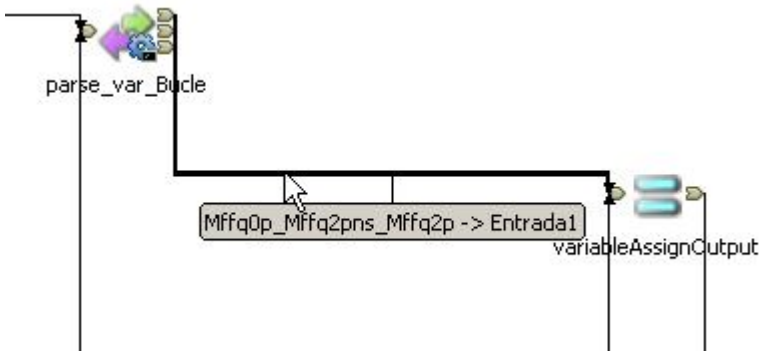


Figura 3.141 Conexión donde vamos a unir el camino que hemos grabado

Hay que tener cuidado de no confundirla con la conexión de la Figura 3.142 que es por donde transcurre el flujo cuando se llega a la pantalla con la descripción HasMsg, cuando en el campo correspondiente aparece el mensaje de FIN DE DATOS, es decir cuando se finaliza el bucle. Para ello nos fijamos en lo que pone en la etiqueta que aparece.

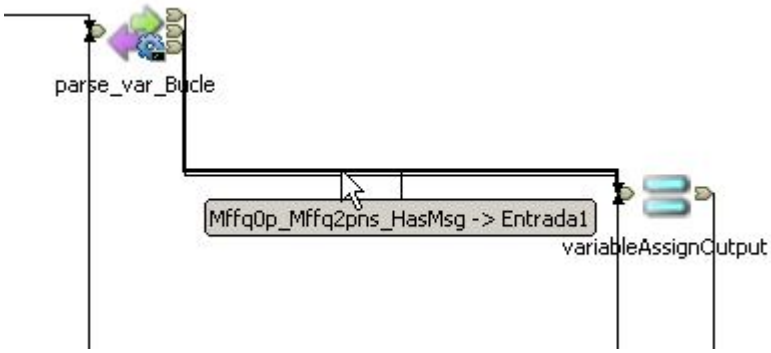


Figura 3.142 Conexión de final de bucle

Seleccionamos la conexión que hemos dicho haciendo clic sorbe ella y vemos que han aparecido unos recuadros al principio y al final de la conexión como podemos ver en la Figura 3.143.

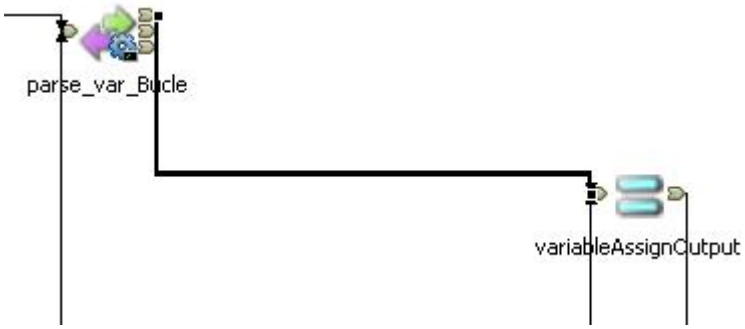


Figura 3.143 Conexión seleccionada

Pinchamos sobre el recuadro del final de la conexión, cuando lo podamos hacer el cursor del ratón cambiará a una +, lo arrastramos sobre la flecha de la izquierda del primer nodo del camino que acabamos de grabar y lo soltamos de forma que queden las conexiones como podemos ver en la Figura 3.144.

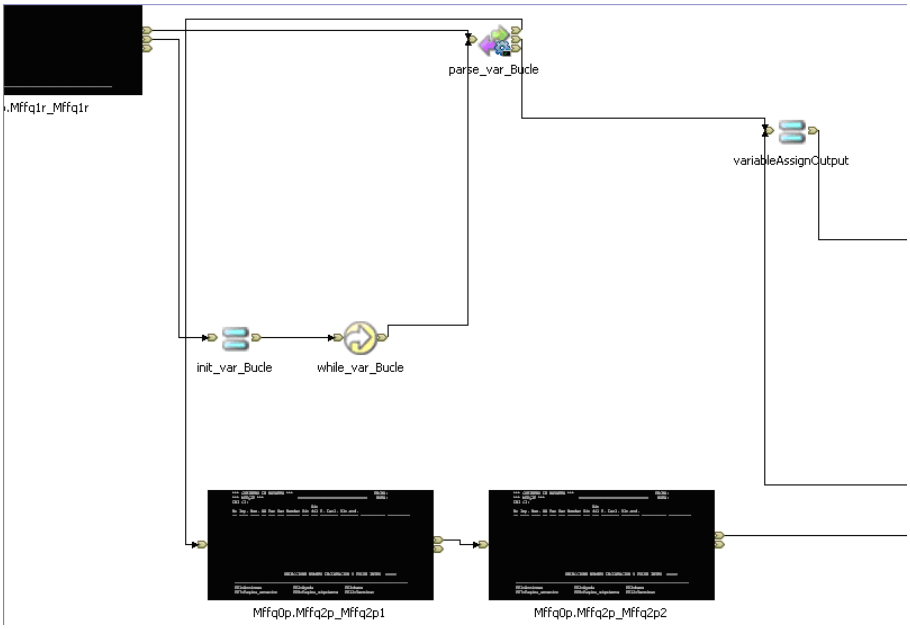


Figura 3.144 Conexión con el nuevo camino establecida

Guardamos los cambios con el botón de la Figura 3.65. Realizamos los mismos pasos pero ahora la pantalla de inicio será la de final de bucle del listado. Esto lo hacemos ya que, si queremos seleccionar un elemento de la última página del listado, la primera pantalla que veremos será esta al finalizar el bucle en una pantalla posterior como ya hemos indicado. Al tener esta pantalla una descripción diferente al del resto de pantallas del listado, el flujo no la reconoce como la primera pantalla del camino de pedir detalles por lo que no funciona correctamente. De esta forma no podemos pedir detalles de un elemento de la pantalla de final de bucle pero no tenemos problemas ya que en esta pantalla nunca hay elementos, solo está el mensaje de "FIN DE DATOS" como podemos ver en a Figura 3.125.

En este caso, el camino grabado lo uniremos con la conexión que se muestra en la Figura 3.145 al contrario de lo que hemos hecho antes, cuando lo hemos unido con la otra conexión.

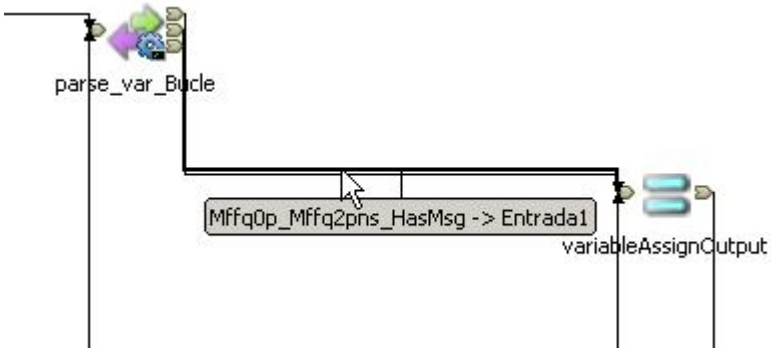


Figura 3.145 Conexión donde unimos el nuevo camino

Una vez realizadas estas operaciones grabamos los cambios por medio del botón de la Figura 3.65.

Una vez guardado el flujo ya podemos generar sus archivos, transferirlos al mainframe e instalar el flujo.

Para probarlo lo tenemos que hacer obligatoriamente por medio del CTG y pasándole un usuario, o usuario y contraseña si así se requiere. Esto hay que hacerlo porque, como hemos visto, esta transacción tiene seguridad. De esta forma la transacción se ejecutará con el usuario que le hayamos pasado en la petición ECI y así evitamos que nos aparezca el mensaje de la Figura. 3.119. Lo probamos pidiendo el mismo elemento pero de diferente página. Por ejemplo el elemento 5 de la página 1 y de la página 2. Vemos como los datos que se obtienen son diferentes en ambas peticiones pero los mismos que si los pedimos por medio de una conexión al CICS por lo que consideramos que el flujo está bien creado. Hay que indicar que la variable de numPaginas la hemos creado de 2 dígitos. Si sólo le pasamos 1, por ejemplo porque queremos la página 5, en la entrada correspondiente tenemos que rellenar la longitud total de la variable con ceros a la izquierda, en este caso 05. Si no hacemos esto la comparación con el índice del bucle no se hace correctamente y el flujo llega hasta la pantalla de final de bucle.

3.8.- Archivos que forman un flujo y borrado de flujos

Una vez que hemos aprendido como grabar flujos, vamos a ver exactamente que archivos se generan y para qué sirve cada uno de estos archivos. También veremos que archivos tenemos que borrar para borrar un flujo en caso de que nos hayamos equivocado al grabarlo o no haga lo que nosotros pretendemos y vayamos a empezar su grabación de cero. Si nos fijamos en el "Explorador de proyectos" nos vamos a encontrar con las carpetas que podemos ver en la Figura

3.146 para cada proyecto que tengamos creado. Todas estas carpetas las tendremos una vez que hayamos grabado y generado los archivos de por lo menos un flujo. En caso de no haber grabado o no haber generado los archivos de un flujo, puede que las carpetas “Flujos”, “Correlación” o “Generación” no estén creadas. Se crearán automáticamente conforme se vayan necesitando.

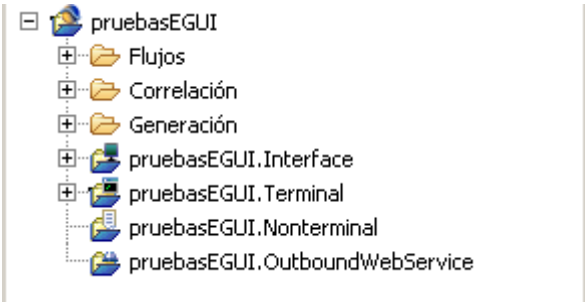


Figura 3.146 Carpetas de un proyecto

En la carpeta “Flujos” se guardan los archivos de los flujos que hemos grabado como podemos ver en la Figura 3.147 donde sólo tenemos el flujo “flujoEGUIconBucle”. Podemos abrir estos flujos para editarlos como hemos visto anteriormente.

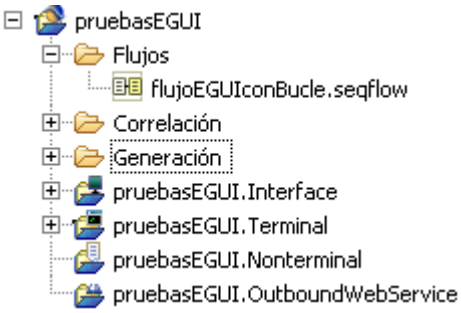


Figura 3.147 Carpeta “Flujos”

En la carpeta “Correlación” se guarda un archivo .sfmap por cada flujo que almacena las asignaciones que se hacen en cada pantalla de los campos de las pantallas a variables de salida o de variables de entrada y datos estáticos a los campos de las pantallas. Esto es necesario para que el flujo sepa las asignaciones que tiene que hacer en cada pantalla. Esto lo podemos ver en la Figura 3.148. En algunos casos abriremos estas correlaciones para editarlas aunque normalmente lo haremos haciendo clic derecho sobre la pantalla que queremos editar y eligiendo “Abrir rutina de correlación” como hemos para crear el bucle con una sola pantalla de listado.

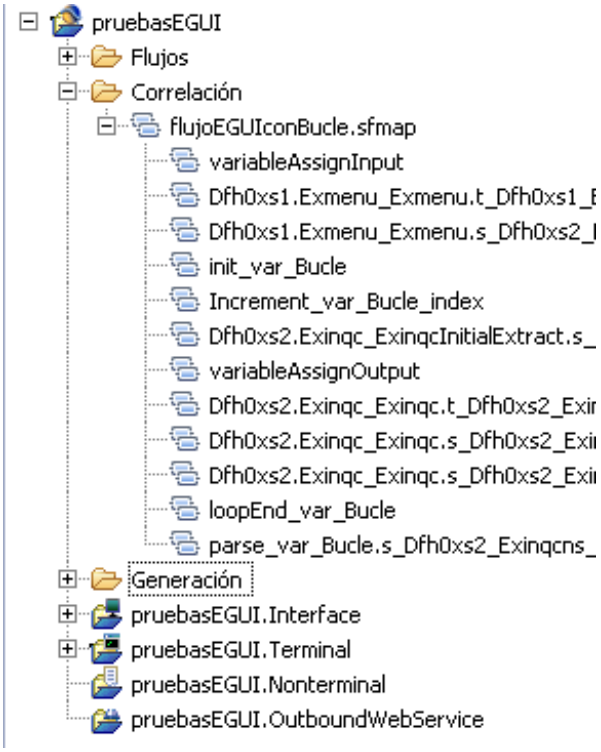
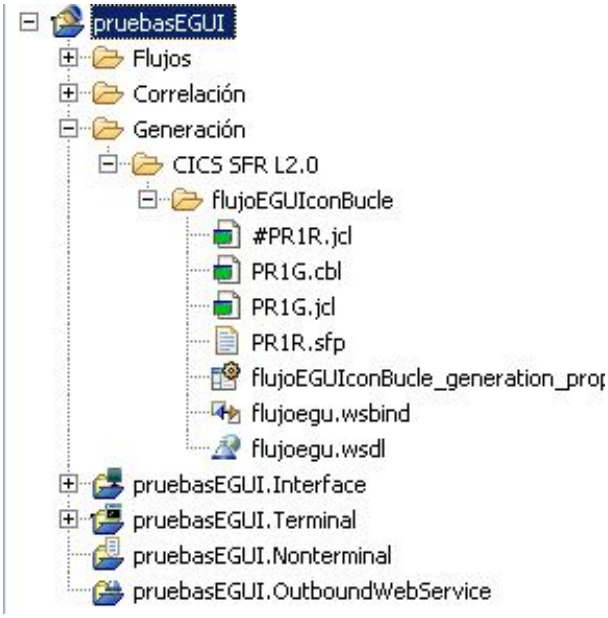


Figura 3.148 Carpeta "Correlación"

Estas dos carpetas se crean cuando empecemos a grabar nuestro primer flujo.

En la carpeta "Generación" se guardan los archivos que se generan para posteriormente transferirlos al mainframe y poder instalar el flujo. Dentro hay una carpeta para las generaciones de Nivel 2.0, que es el que utilizamos, y dentro se crea una carpeta por cada flujo donde se guardan los archivos generados para ese flujo como podemos ver en la Figura 3.149. Normalmente nos vamos a editar directamente ninguno de los archivos que contiene, mas que el de propiedades de generación del flujo.



3.149 Carpeta "Generación"

Dentro de la carpeta de cada flujo hay los siguientes archivos con la utilidad que indicamos:

- Archivo #xxx.jcl
  - JCl que crea las definiciones de recursos necesarias para que el flujo funcione
- Arhivo xxx.cbl
  - Archivo COBOL que es el fuente del programa que se ejecuta al invocar el flujo
- Archivo xxx.jcl
  - JCL que compilado el programa COBOL anterior
- Arhcivo xxx.sfp
  - Archivo de propiedades del flujo que lee la transacción CMAN para instalar el flujo
- Arhico xx.sfgen
  - Archio de propiedades de generación del flujo
- Archivos xxx.wsbin y xxx.wsdl
  - Archivos necesarios para poder invocar el flujo desde una llamada a un servicio Web

En la carpeta "*nombre_proyecto*.Interface" tenemos dos carpetas llamadas "Operaciones" y "Mensajes" como podemos ver en la Figura 3.150.

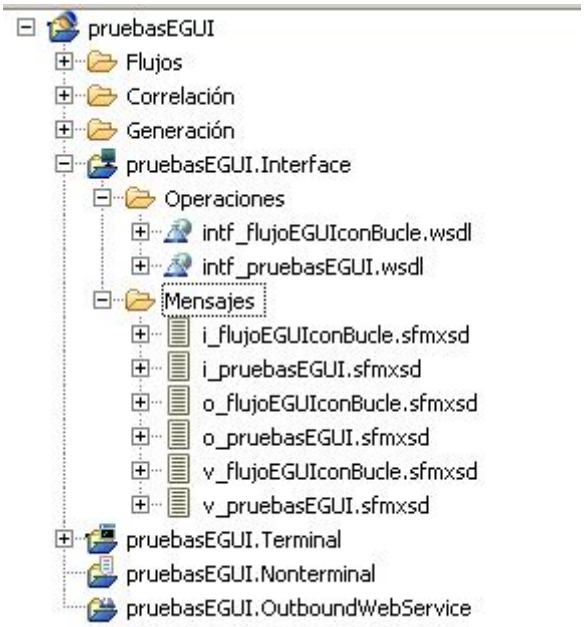


Figura 3.150 Carpeta "*nombre_proyecto*.Interface

En la carpeta "Operaciones" tenemos un archivo wsdl por cada flujo y uno para el proyecto que almacena las operaciones que se pueden invocar sobre cada flujo o proyecto. En general, si es un flujo, sólo tendrá una operación que sirve para invocar al flujo por medio de un servicio Web. No vamos a editar directamente estos archivos.

En la carpeta "Mensajes" tenemos los grupos de variables de entrada, auxiliares y de salida de cada flujo. En general, si dejamos que RDz asigne los nombres de los grupos automáticamente, los nombre de los grupos serán "i_*nombreFlujo*" para el grupo de variables de entrada, "v_*nombreFlujo*" para las variables auxiliares y "o_*nombreFlujo*" para las variables de salida. En algunos casos editaremos estos grupos de variables, normalmente para añadir alguna.

En la carpeta "*nombre_proyecto*.Terminal" vamos a tener las carpetas "Hosts", "Operaciones" y "Mensajes" como podemos ver en la Figura 3.151.

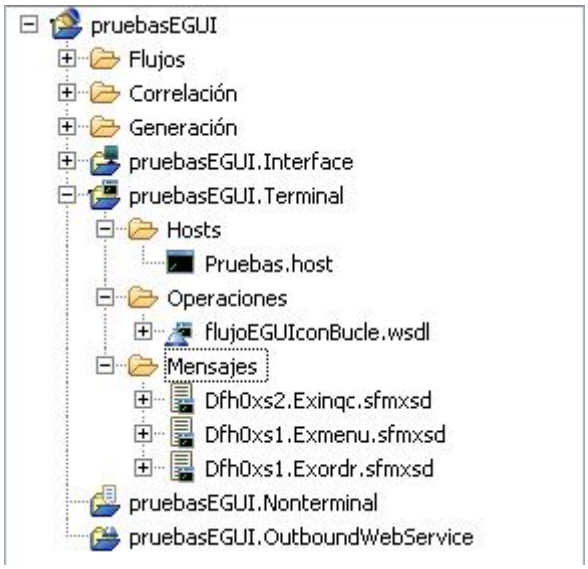


Figura 3.151 Carpeta "*nombre_proyecto*.Interface"

La carpeta "Hosts" contiene las conexiones que hayamos creado con el mainframe.

La carpeta "Operaciones" contiene un archivo wsdl donde se van almacenando las diferentes operaciones que realizamos sobre cada pantalla del flujo. Es similar a la carpeta "Correlación" aunque la información que aquí aparece no se puede editar.

La carpeta "Mensajes" contiene los mapas de las pantallas ya sean las que hemos importado su mapa BMS o las que hemos capturado por medio del RDz. Estos archivos tampoco los vamos a editar.

Las carpetas "*nombre_proyecto*.Noterminal" y "*nombre_proyecto*.OutboundWerService" no van a contener ningún archivo si realizamos las tareas que hemos visto con el RDz.

Si por lo que sea queremos borrar un flujo, ya sea porque nos hemos equivocado al grabarlo o por cualquier otra razón, no lo podemos eliminar automáticamente. Tenemos que borrar los siguientes archivos correspondientes al flujo que queremos borrar:

- Archivo seqflow de la carpeta "Flujos"
- Archivo sfmap de la carpeta "Correlación"
- Carpeta "*nombreFlujo*" dentro del Nivel 2.0 de la carpeta "Generación"
- Archivo wsdl de la carpeta "*nombre_proyecto*.Interface → Operaciones"
- Archivos sfmxsd de la carpeta "*nombre_proyecto*.Interface → Mensajes"
- Archivo wsdl de la carpeta "*nombre_proyecto*.Terminal → Operaciones"
- Si queremos borrar algún mapa de pantalla, los archivos sfmxsd de la carpeta "*nombre_proyecto*.Terminal → Mensajes"

3.9.- Tiempo de ejecución de flujos

Para comprobar el tiempo que tarda en ejecutarse un flujo realizando su invocación por medio del CTG, insertamos temporizadores antes de enviar la petición y después de obtener respuesta. Para un flujo como el último, donde se recorren dos pantallas de introducción de datos, una

pantalla informativa, varias pantallas de listado y por último una pantalla de donde obtenemos los resultados de salida, el tiempo de ejecución de este flujo, es de aproximadamente medio segundo. Si a esto le añadimos el tiempo que nos cuesta introducir los datos de entrada nos cuesta obtener los datos que queremos es de aproximadamente 4 segundos.

Este tiempo es menor al que nos cuesta navegar hasta la pantalla de listado, buscar la página donde se encuentra el elemento del que queremos pedir los detalles, aun sabiendo en que página se encuentra de antemano, y pedir sus detalles.

Esta diferencia se hace mayor cuantas más pantallas tenemos que recorrer introduciendo datos estáticos. Es decir, datos que nunca cambian sino que los necesitaos introducir para poder ejecutar la tarea que deseemos.

Esto hace que el uso del SFF, además de permitir una integración de aplicaciones conversacionales o pseudo-conversacionales con sistemas externos, y de modernizar la forma de comunicarse con ellas, también puede mejorar el rendimiento de interactuar con estas aplicaciones.

4.- Conclusiones

4.1.- Logros en los objetivos del proyecto

Los objetivos fundamentales eran lograr modernizar las aplicaciones de los mainframes, en especial del CICS, y lograr la interoperabilidad e integración de estas aplicaciones con otros sistemas de información externos como pueden ser aplicaciones escritas en Java o en .Net. Por modernizar entendemos el uso de elementos actuales como el ratón o los atajos de teclado para poder interactuar con las aplicaciones, cosa que el CICS no permite.

Hemos visto como por medio del CICS Transaction Gateway, o CTG, y de las librerías y conectores que provee, podemos llamar a transacciones no conversacionales. Estas transacciones son las que no utilizan pantallas para comunicarse con el usuario sino que lo hacen por medio de un área de intercambio de datos, ya sea una COMMAREA o Canales y Contenedores. Hemos logrado ejecutar correctamente este tipo de transacciones, y los programas que ellas invocan, desde aplicaciones escritas en Java y en VB.Net. También se puede llamar a este tipo de transacciones desde lenguajes C, C++ o COBOL. Aunque las llamadas desde este tipo de aplicaciones no se han probado, la API utilizada es la misma que la de VB.Net por lo que podemos suponer que funciona correctamente.

Por medio del CICS Service Flow Feature, CSFF o SFF, hemos logrado crear unos flujos que, al ser llamados desde el CTG, o desde servicios Web, permiten ejecutar transacciones conversacionales o pseudo-conversacionales. Estas transacciones son las que utilizan pantallas para comunicarse con el usuario, ya sea para mostrarle datos o para pedírselos. Hemos logrado llamar a estos flujos, y por lo tanto a las transacciones, también desde aplicaciones Java y VB.Net.

Al lograr llamar a transacciones conversacionales y no conversacionales desde Java y .Net, logramos evidentemente integrar las aplicaciones del CICS en sistemas de información externos desarrollados en estos lenguajes. Además, por medio de las interfaces de usuario que podemos desarrollar por medio de estos lenguajes, logramos modernizar estas aplicaciones haciendo que la interacción con ellas sea más “moderno” al permitir el uso del ratón, atajos de teclado para copiar o pegar datos, etc.

Considero que por medio de estos dos productos, el CTG y el SFF, y de las características que ofrecen, se ha logrado cumplir los objetivos que se planteaban en este proyecto.

4.2.- Vías futuras de trabajo

Debido a la arquitectura y limitaciones del sistema que se tiene instalado, no se han podido probar todas las características que ofrecen tanto el CTG como el SFF. Estas características se pueden probar en futuros proyectos que tomen este como punto de partida. Especialmente es conveniente probar las siguientes características del CTG:

- API de programación ESI y EPI
  - Para ello el CTG tiene que estar instalado en la misma máquina que el cliente que hace las peticiones ya sea sobre Windows, Unix o Linux.
- Probar las transacciones XA y el protocolo two-phase commit
- Si se tiene la versión 8.0 o superior del CTG, probar las llamadas a programas de Canales y Contenedores desde C/C++, .Net o similar

Anexo 1.- Uso del BlueZone FTP

El BlueZone FTP nos va a permitir transferir archivos entre nuestra máquina y el mainframe, ya sea en modo binario o en modo texto. Para ello iniciamos el BlueZone Secure FTP. Nos aparecerá la pantalla de la Figura A1.1 donde podemos crear o seleccionar un perfil para almacenar nuestras conexiones y preferencias. Creamos un perfil llamado “pruebas”.

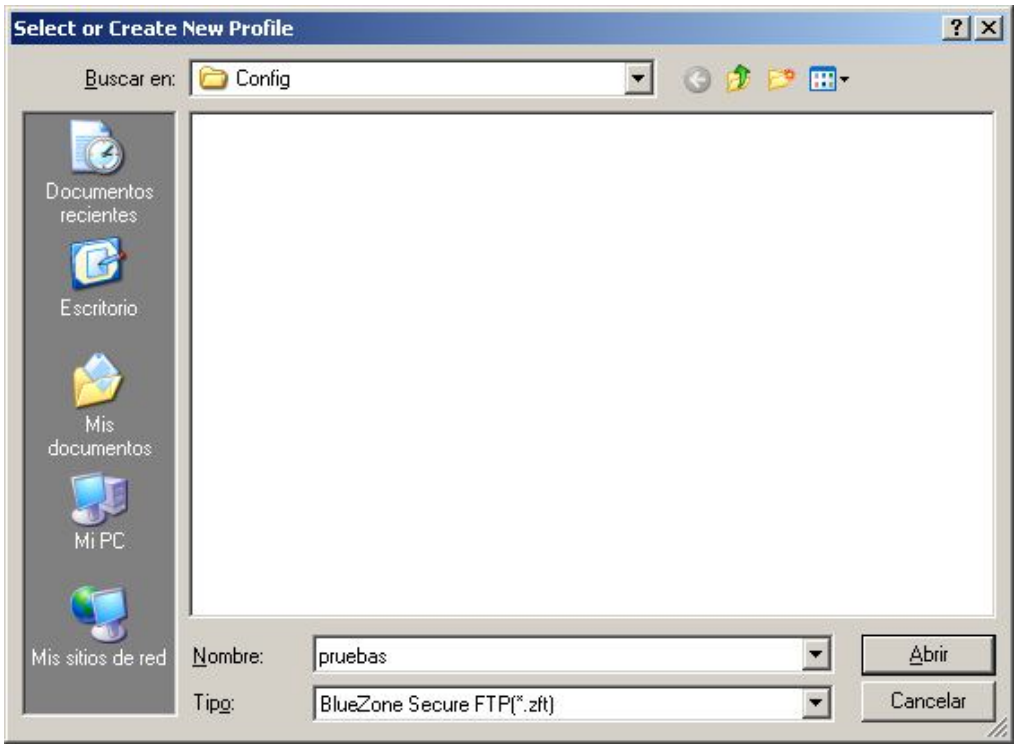


Figura A1.1 Pantalla de selección de perfil

Cuando pulsemos abrir nos aparecerá un mensaje indicándonos que el perfil no existe y si lo queremos crear. Le decimos que si. Automáticamente nos aparecerá la ventana de la Figura A1.2 para que podamos configurar una conexión con el mainframe. La configuramos como se puede ver.

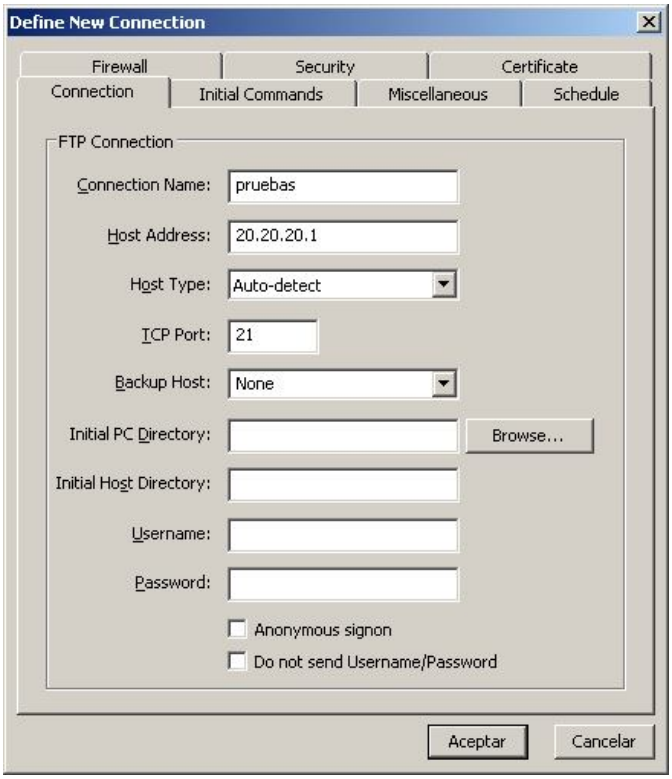


Figura A1.2 Pantalla de creación de conexión

Pulsamos "Aceptar" y nos aparece la ventana de la Figura A1.3 para que podamos seleccionar la conexión que queremos abrir.

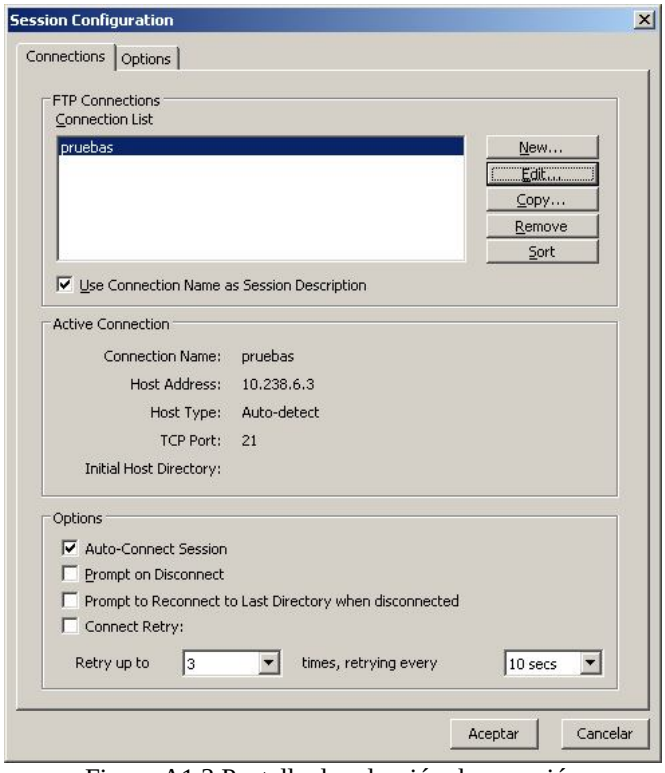


Figura A1.3 Pantalla de selección de conexión

Seleccionamos la conexión que acabamos de crear y pulsamos “Aceptar”. En la pantalla principal del BlueZone, pulsamos el botón que podemos ver en la Figura A1.4 para iniciar la conexión.

Figura A1.4 Botón de conexión

Nos aparecerá una ventana para que introduzcamos nuestro usuario de RACF y a continuación otro para que introduzcamos nuestra contraseña. Cuando compruebe que ambos son válidos nos conectará al mainframe. Una vez hecho esto se nos presentará la pantalla principal del BlueZone que podemos ver en la Figura A1.5.

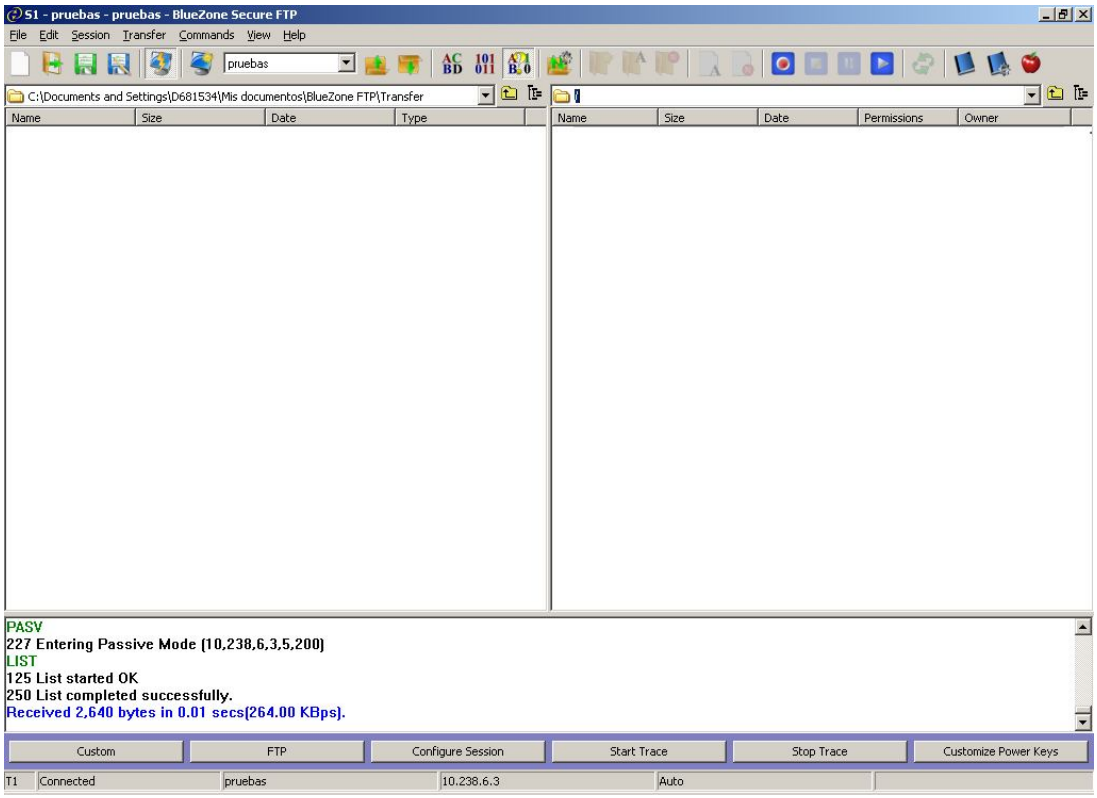


Figura A1.5 Pantalla principal de BlueZone

En la parte izquierda tenemos un explorador de archivos para nuestra máquina mientras que en la parte derecha tenemos el del mainframe. En la parte mainframe podemos acceder tanto a data sets como a ficheros y archivos de la parte Unix.

Para transferir un archivo lo seleccionamos en la parte correspondiente y en la otra seleccionamos la carpeta, o el data set si es el mainframe, donde lo queremos transferir. Una vez hecho esto seleccionamos en la parte superior el botón correspondiente para indicar si lo queremos transferir en modo binario o en modo texto (ASCII). En la Figura A1.6 podemos ver respectivamente los botones para seleccionar la transferencia en modo binario o texto.

Figura A1.6 Transferencia en modo binario y modo texto respectivamente

Existe un tercer botón para ponerlo en modo automático y que el BlueZone decida como transferir el archivo pero no lo vamos a utilizar.

Una vez seleccionado el modo de transferencia pulsamos uno de los botones de la Figura A1.7 para transferir respectivamente el archivo del mainframe a nuestra máquina, a lo que llama recibir archivos, o viceversa, enviar archivos.

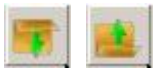


Figura A1.7 Botón para transferir archivos del mainframe a nuestra máquina o viceversa

Cuando realicemos alguna transferencia, o cualquier otra operación sobre el mainframe como explorar sus archivos y data sets, en la parte inferior de la pantalla del BlueZone nos aparecerá en un Log las operaciones que se están realizando y si se está produciendo algún error. Por ejemplo, en la Figura A1.8 podemos ver este Log indicándonos que se ha transferido correctamente el archivo /SYSTEM/etc/ctgvar/ctg.ini del mainframe a nuestra máquina.

Figura A1.8 Transferencia correcta del archivo ctg.ini a nuestra máquina

Con esto ya tendríamos todo lo necesario para poder transferir archivos entre nuestra máquina y el mainframe.

Bibliografía

- CICS TRANSACTION GATEWAY FOR z/OS VERSION 6.1. Nigel Williams, Colin Alcock, Steven Day, Tommy Joergensen, Rob Jones & Phil Wakelin. 2006
- SERVICE FLOW FEATURE OF CICS TRANSACTION SERVER FOR z/OS, VERSION 3.2. Nick Carlin & William Alexander 2008
- USING THE CICS SERVICE FLOW FEATURE. Scott Clee 2007
- z/OS TSO/E USER'S GUIDE. 2004
- z/OS UNIX SYSTEM SERVICE USER'S GUIDE. 2007
- ISPF USER'S GUIDE VOLUME 1 z/OS VERSION 1 RELEASE 8.0. 2006
- ISPF USER'S GUIDE VOLUME 2 z/OS VERSION 1 RELEASE 8.0. 2006
- CICS SERVICE FLOW RUNTIME USER'S GUIDE VERSION 3.2. 2010
- CICS TRANSACTION GATEWAY Z/OS ADMINISTRATION VERSION 7.2. 2008
- CICS TRANSACTION GATEWAY PROGRAMMING GUIDE VERSION 7.2. 2008
- IBM RATIONAL DEVELOPER FOR SYSTEM Z VERSION 7.6 - ENTERPRISE SERVICE TOOLS FOR WEB SERVICES AND SOA SERVICE FLOW PROJECTS. 2009
- INFOCENTER CICS TS VERSION 3.2 - HTTP://PUBLIB.BOULDER.IBM.COM/INFOCENTER/CICSTS/V3R2/INDEX.JSP
- INFOCENTER CICS TG VERSION 7.2 - HTTP://PUBLIB.BOULDER.IBM.COM/INFOCENTER/CICSTGZO/V7R2/INDEX.JSP