UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TECNICA SUPERIOR DE INGENIERIA INFORMATICA

Proyecto de Fin de Carrera de Ingeniero Informático

**Analyzer & Profiler for COBOL Batch**

FRANCISCO JAVIER GONZÁLEZ GRÁNDEZ

Dirigido por: NOMBRE DEL DIRECTOR

Supervisado por: JAVIER VÉLEZ

Curso: CURSO (convocatoria de defensa)





**Analyzer & Profiler for COBOL Batch**

Proyecto de Fin de Carrera de modalidad *oferta específica*

Realizado por: FRANCISCO JAVIER GONZÁLEZ GRÁNDEZ (firma)

Dirigido por: NOMBRE DEL DIRECTOR (firma)

Supervisado por: JAVIER VÉLEZ (firma)

Tribunal calificador:

Presidente: D./Dª

(firma)

Secretario: D./Dª

(firma)

Vocal: D./Dª

(firma)

Fecha de lectura y defensa:

Calificación:

**ÍNDICE**

Siglas, abreviaturas y acrónimos 8

Definiciones 9

1 Contexto 11

2 Planteamiento de la solución 13

2.1 Análisis de la calidad del código 13

2.2 Análisis del comportamiento 14

2.3 Ámbitos de aplicación 15

2.4 Análisis económico 16

3 Conceptos 19

3.1 COBOL 19

3.1.1 Estructura 19

3.1.2 Variables 20

3.1.3 Flujo de ejecución 22

3.1.4 Rutinas 23

3.2 Complejidad ciclomática 26

3.3 Tiempos y Consumos 29

4 Visión general 33

4.1 Arquitectura 33

5 Parser 34

6 Trapper 35

6.1 Consideraciones de diseño 35

6.1.1 Overhead (Sobrecarga) 35

6.1.2 Flujo de ejecución 36

6.1.3 Generación óptima de traps 38

6.1.4 Generación óptima de mensajes 40

6.2 Implementación 41

6.2.1 Diagrama de flujo 41

6.3 Arquitectura 43

6.3.1 Ficheros fuente 44

6.3.2 Ficheros de cabecera 44

6.3.3 Dependencias externas 45

6.3.4 Estructuras de datos 45

6.4 Mensajes 45

6.4.1 Inicio de sesión 45

6.4.2 Fin de sesión 45

6.4.3 Módulo 46

6.4.4 Párrafo 46

6.4.5 Call 47

6.4.6 Cobertura 47

6.4.7 Ficheros 47

6.4.8 Perform 48

7 Collector 49

7.1 Insercion de variables 49

7.1.1 Collector / Parser 50

7.1.2 Collector / Trapper 50

7.1.3 Collector / Consolidator 50

8 Modelo de datos 51

8.1 Visión general 51

8.2 Tablas generales 53

8.2.1 CFG\_CONFIGURACION 53

8.3 Tablas base 55

8.3.1 SDP\_APLICACION 56

8.3.2 SDP\_REL\_APP\_MOD 57

8.4 Tablas de módulo 59

8.5 Tablas de ejecución 61

8.6 Tablas de sesión 63

8.7 Tablas sumarizadas 65

8.8 Tablas de *Logging* 66

9 Instalación y configuración 69

9.1 Arquitectura hardware 69

9.2 Arquitectura Software 70

9.3 Servidor 71

9.3.1 Sistema de mensajería WebSphere MQ Series 72

9.4 Configuración 72

9.4.1 Servidor 72

9.4.1.1 Usuarios 72

9.4.2 MQSeries 73

9.4.2.1 MQSeries 74

9.4.2.2 Establecer la instalación principal 74

9.4.2.3 Creación del gestor de colas 74

9.4.2.4 Creación de los objetos 74

9.4.2.5 Establecimiento de los permisos 75

9.4.2.6 Automatización del arranque 75

9.4.3 Servidores de páginas y de aplicaciones 76

9.4.3.1 Configurar el servidor de aplicaciones 76

9.4.3.2 Configurar el servidor de páginas 77

9.4.4 Servidor de base de datos 78

10 Manual de usuario 79

10.1 Parámetros y configuración 79

10.1.1 Fichero de configuración 79

10.1.2 Variables de entorno 80

10.2 SDPParser 81

**Índice de figuras**

Diagrama 1: Árbol llamadas estáticas 25

Diagrama 2: Grafos de control de flujo 27

Diagrama 3: Grafo condensado vs grafo expandido 28

Diagrama 4: Grafo con varios bloques 29

Diagrama 5: Ejemplo de pila de monitorización 37

Diagrama 6: Relación de las estructuras de SDPTRAPB 41

Diagrama 7: Flujo principal SDPTRAPB 42

Diagrama 8: Flujo de mensaje de fin 43

Diagrama 9: Arquitectura SDPTRAPB 43

Diagrama 10: Relación entre grupos de tablas 52

Diagrama 11: E/R Grupo tablas SDP 55

Diagrama 12: E/R Grupo tablas MOD 60

Diagrama 13: E/R Grupo de tablas TRP 62

Diagrama 14: E/R Grupo tablas SES 64

Diagrama 15: E/R Grupo de tablas SUM 66

Diagrama 16: E/R tablas grupo LOG 68

Diagrama 17: Arquitectura Hardware 69

Diagrama 18: Arquitectura Software 70

Código 1: DOCB0001 - Declaración de variables 21

Código 2: DOCB0101 23

Código 3: DOCB0102 23

Código 4: DOCB0201 23

Código 5: Captura de tiempos 37

Código 6: Ordenación método de la burbuja 39

Código 7: Ejemplo de fichero workers.properties 76

Código 8: Ejemplo tomcat-users.xml 77

Código 9: Virtual Host SDP.conf 77

Código 10: Autorizar usuario en MySQL 78

Tabla 1: Listado de módulos fuente SDPTRAPB 45

Tabla 2: Listado de ficheros de cabecera SDPTRAPB 46

Tabla 3: Formato mensaje BEG\_SESION 46

Tabla 4: Formato mensaje END\_SESION 47

Tabla 5: : Formato mensaje BEG\_MODULE 47

Tabla 6: Formato mensaje BEG\_PARAGRAPH 48

Tabla 7: Formato mensaje BEG\_CALL 48

Tabla 8: Formato mensaje MSG\_COVER 48

Tabla 9: : Formato mensaje MSG\_PERSISTENCE 49

Tabla 10: Formato mensaje MSG\_PERFORM 49

Tabla 11: Prefijos de tablas 52

Tabla 12: Tablas generales 54

Tabla 13: Tabla CFG\_CONFIGURATION 55

Tabla 14: Tablas SDP 56

Tabla 15:Tabla SDP\_APLICACION 57

Tabla 16: Tabla SDP\_REL\_APP\_MOD 58

Tabla 17: Tablas MOD 60

Tabla 18: Tablas TRP 62

Tabla 19: Tablas SES 64

Tabla 20: Tablas SUM 66

Tabla 21: Tablas LOG 68

Tabla 22: Componentes software 72

Tabla 23: Paquetes a instalar 72

Tabla 24: Paquetes MQ Series a instalar 73

Tabla 25: Usuarios servidor 74

Tabla 26: Objetos de mensajeria 74

Siglas, abreviaturas y acrónimos

|  |  |
| --- | --- |
| Concepto | Definición |
|  |  |
| API | *Application Program Interface*  Interfaz de definición de la funciones disponibles por un componente software |
|  |  |
| JMS | *Java Message Service*  Especificación del sistema de mensajería basado en Java |
|  |  |
| MIPS | Millones de Instrucciones por Segundo. En plataformas IBM es una unidad de medida del consumo de CPU |
|  |  |
| PID | Process Indetification  Identificador numérico dado por el Sistema Operativo a un proceso cuando se está ejecutando |
|  |  |
| PYME | Pequeña y mediana empresa |
|  |  |
| Q&A | *Quality & Assurance*  En este documento hace referencia al departamento responsable de establecer y mantener los procedimientos y controles de calidad del software |
|  |  |
| SGDB | Sistema Gestor de Bases de Datos. |
|  |  |
| TIC | Tecnologías de la información y las comunicaciones. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Definiciones

|  |  |
| --- | --- |
| Aplicación | Conjunto de programas y módulos que implementan un conjunto de funcionalidades de negocio |
|  |  |
| Área | Conjunto de aplicaciones |
|  |  |
| Back-end | Componente o componentes de la aplicación o del sistema que no son accesibles por el usuario final. Normalmente procesos Batch, aunque también se incluye aquí las posibles interfaces de usuario (pantallas, páginas Web, etc.) que son utilizadas por los usuarios de la aplicación |
|  |  |
| *Front-end* | Componente o componentes de la aplicación o del sistema con los que interactúa el usuario final. Normalmente pantallas o aplicaciones Web |
|  |  |
| Middleware | Componente software que se encarga de comunicar o integrar el front-end y el back-end de un sistema |
|  |  |
| Módulo | Unidad de código ejecutable que forma parte de un programa o que es invocado por este |
|  |  |
| *Porting* | Acción de tomar un código fuente y convertirlo en otro, bien sea en el mismo lenguaje pero capaz de ejecutarse en una máquina diferente o bien convirtiéndolo a otro lenguaje de programación |
|  |  |
| Programa | Unidad de código ejecutable que puede ser invocado directamente por el usuario o por un proceso |
|  |  |
| Rutina | Unidad de código ejecutable que forma parte de un programa o que es invocado por este |
|  |  |
| Sesión | Unidad ejecutable |
|  |  |
| Sistema | Conjunto de aplicaciones, productos software y componentes hardware que funcionan de manera colaborativa para proveer determinados servicios y/o funcionalidades |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Usuario final | Persona física o componente software que hace uso de las funcionalidades de negocio provistas por la aplicación |
|  |  |
| Usuario | Persona física o componente software que hace uso de las funcionalidades de la aplicación, en contraposición de usuario final, este usuario utiliza funcionalidades que no son de negocio. Por ejemplo, el administrador del sistema es un usuario del mismo, pero no es un usuario final |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Contexto

Actualmente, cuando se habla de Sistemas de Información y de lenguajes de programación, resulta normal pensar en lenguajes de programación orientados a objetos como Java, interpretados como Javascript, PHP, Perl, y a veces, pero no con tanta frecuencia, C y C++, como indica, por ejemplo, el informe Introducing the Gartner Programming Language Index for 2014 [GAR1] o el Indice TIOBE [TIO1].

Sin embargo, tal vez no somos conscientes de que según algunos estudios realizados a lo largo del año 2.013 por empresas como por ejemplo, MicroFocus, Gartner, IT Media o ComputerPro [CP1] entre otros, reflejan otro tipo de información, como por ejemplo:

* Durante el año 2013 se crearon más programas COBOL que páginas Web
* COBOL soporta cada día el 90% de los sistemas de negocio de Fortune 500
* COBOL soporta el 70% de la lógica de negocio crítica
* COBOL procesa el 85% de las transacciones comerciales diarias
* Un ciudadano interactúa una media de 10 veces al día con un programa COBOL

Lo cual da una idea de la relevancia que todavía mantiene este lenguaje de programación.

Por otro lado, y a diferencia de otros tipos de aplicaciones desarrolladas en otros lenguajes, suele ser bastante habitual que los sistemas hardware/software sobre los que se ejecutan estas aplicaciones tengan un coste de explotación variable en función del uso de los recursos consumidos o necesitados (MIPS, Memoria, Procesadores, etc.) para su ejecución, por lo que el coste asociado a la generación de un programa COBOL no se limita, simplificando, al coste de su codificación, pruebas y mantenimiento, si no que se extiende a lo largo de todo su ciclo de vida y supone un coste para la empresa propietaria del programa cada vez que este es ejecutado.

Por lo tanto, los conceptos de corrección y eficiencia de un programa asociados al desarrollo de aplicaciones, y obviados tan a menudo, en aras de la rapidez, economía y reducción de los plazos de entrega, cobran en este entorno una importancia mucho mayor de la que ya de por sí debería tener para cualquier sistema empresarial ya que cada vez que se ejecuta un programa COBOL no óptimo, este repercute directamente en la cuenta de resultados de la empresa.

A pesar de todo ello, y a diferencia de otros lenguajes como C o Java, prácticamente no existen herramientas que permitan realizar un control de calidad del código desarrollado antes de ser ejecutado, y los que hay son bastantes limitados y con un fuerte componente manual.

Por mencionar solo un ejemplo, sin embargo muy habitual en este entorno, el método de validación de la corrección de un programa en una de las primeras entidades financieras del mundo, se basa en entregar a un departamento de control de calidad un listado en papel con los resultados de las pruebas realizadas.

Esta situación se vuelve más crítica cuando se trata de evaluar el rendimiento y el comportamiento de los programas en entornos productivos, puesto que los administradores de los sistemas no tienen la información necesaria para evaluar si un programa se está comportando adecuadamente y las herramientas disponibles no están orientadas al análisis de programas en concreto sino más bien al comportamiento del sistema y no están disponibles ni para los equipos de desarrollo ni para los departamentos responsables

# Planteamiento de la solución

Este proyecto presenta una solución a los problemas descritos en el apartado anterior utilizando como dialecto del lenguaje la implementación desarrollada por el proyecto open-source GNUCobol [GNUC] abordando las dos perspectivas:

1. Análisis de la calidad del código fuente desarrollado
2. Análisis del comportamiento del programa en un entorno productivo

## Análisis de la calidad del código

El enfoque utilizado para realizar este proceso se basa en “pseudo-compilar” el código fuente utilizando las herramientas de análisis léxico, sintáctico y semántico típicas en el desarrollo de compiladores para construir los arboles semánticos correspondientes, los grafos de ejecución y extraer la información relevante que permita analizar la complejidad del código desarrollado, su adecuación a las directivas de desarrollo definidas por la entidad para la cual se ha desarrollado y su adecuación a unas métricas de calidad del software homogéneas.

Para clarificar este proceso XXXXX

Sentencias no permitidas

El ejemplo más evidente sería la sentencia GO TO, que, como la propia sentencia indica, realiza un salto incondicional a otra parte del código. Esta sentencia se sigue manteniendo en todas las versiones del lenguaje, pero está prohibido su uso en todas las instalaciones de Sistemas de Información.

Al ejecutar el “pseudo-compilador” sobre un código que tuviera esta sentencia, y de acuerdo con las normas definidas en la instalación, el sistema generará un error indicando que existen instrucciones no permitidas, impidiendo que el programa se compile correctamente.

Análisis de la legibilidad

El sistema puede avisar de situaciones, que aunque correctas desde el punto de corrección del programa, no se consideran adecuadas:

|  |  |
| --- | --- |
| MOVE  UNA-VARIABLE  TO  OTRA-VARIABLE | MOVE UNA-VARIABLE TO OTRA-VARIABLE |
|  |  |
| IF A THEN  MOVE A TO B  IF B THEN  MOVE B TO C  IF C THEN  MOVE C TO D. | IF A THEN  MOVE A TO B  IF B THEN  MOVE B TO C  IF C THEN  MOVE C TO D. |

Análisis de la complejidad

El sistema calcula la complejidad ciclomática de sus componentes con lo que, se puede evitar que existan bloques de código demasiado complejos que deberían ser refactorizados en bloques más sencillos y permite establecer el número de casos de prueba necesarios para garantizar la corrección del programa.

## Análisis del comportamiento

Para poder realizar el análisis del comportamiento real del programa en un entorno productivo el enfoque utilizado es, basándonos en el mismo proceso de pseudo-compilación, inyectar código especial en los puntos críticos del código fuente que permita monitorizar en tiempo de ejecución el flujo real del programa y los consumos del mismo, a este código especial lo denominaremos TRAP.

Es decir, cuando se procesa un código fuente con el sistema el resultado es **otro código fuente** funcionalmente equivalente al original pero que va aportando información acerca de las partes de código ejecutadas realmente y de sus consumos.

En el momento de diseñar este componente resulta crítico considerar que:

* La inclusión de Traps en el código hace que este tenga que ejecutar más código, lo cual, inevitablemente produce un consumo más de máquina y un tiempo de ejecución superior.
* Como ya se ha indicado en el apartado anterior, la ejecución de un programa tiene un coste monetario dependiente del consumo de máquina realizado,

Para minimizar el impacto en estos dos aspectos:

* El componente de captura de la información se ha desarrollado en C, como lenguaje más óptimo desde el punto de vista de la máquina después del ensamblador, y se ha minimizado su proceso de forma que únicamente sea responsable de capturar la información relevante.
* La información recolectada es enviada para su tratamiento y análisis de la información a un sistema externo, no productivo y cuyos costes no dependan del uso de que se haga del mismo.

## Ámbitos de aplicación

Dado que es un sistema que realiza análisis de programas COBOL, su ámbito de aplicación es, lógicamente, cualquier instalación que tenga software desarrollado en ese lenguaje. Esto supone, simplemente en España, entre otros:

* Todas las entidades financieras de Banca Retail.
* Todas las compañías de telecomunicaciones
* La mayoría de las empresas energéticas
* Transportes: Iberia, Renfe, Adif
* Grandes empresas de servicios y distribución

Dentro de estas empresas determinadas áreas o departamentos que requieran de herramientas de control del software, como por ejemplo:

* Departamento de Q&A: Como herramienta central para establecer los criterios y controles de calidad del software desarrollado.
* Departamentos de operaciones y explotación: Como herramienta para analizar problemas en tiempo de ejecución.
* Departamentos de desarrollo: Como herramienta de control de calidad y de perfilado de programas
* Departamentos operativos que soporten en su contabilidad interna los costes de ejecución de su software o que necesiten herramientas de análisis para prever los recursos necesarios en TIC a medio plazo

Por otro lado, y basándonos en la tecnología de inyección de código, la herramienta se puede utilizar como base para:

* Proyectos de porting de software
* Proyectos de optimización de código
* Proyectos de análisis de calidad del software
* En general, cualquier proyecto que requiera un análisis sintáctico o semántico de código fuente

## Análisis económico

A diferencia de otros productos o sistemas orientados a usuarios finales donde el precio se suele establecer en base a criterios de costes de producción, explotación y volumen de ventas esperado, en este tipo de productos o sistemas orientados principalmente a empresas el coste se suele establecer atendiendo a otros criterios como son:

* **Características de la empresa**: no se suelen aplicar los mismos criterios de precios a una pyme que a una gran empresa.
* **Valor añadido**: O dicho de otra manera, se da mucho más importancia al concepto valor que al concepto precio, en el sentido de que valor añadido o que soluciones aporta el sistema a los posibles problemas o carencias que pueda tener el cliente
* **Usuarios**: Volumen de usuarios que van a utilizar el sistema
* **Máquinas y capacidad de proceso**: Es un procedimiento estándar que el precio de un determinado producto software varíe en función del tipo de máquina donde se vaya a instalar y de la capacidad de proceso de la misma
* **Servicios**: El software se distribuye o necesita de un paquete de servicios profesionales asociados
* **Servicios de consultoría**: Como se ha indicado, el sistema presentado en este proyecto, además de implementar una solución concreta para el problema del análisis de la calidad de código, ofrece una tecnología para la realización de otra serie de proyectos de asesoría, consultoría u otros servicios
* Métodos de facturación no convencionales: Por métodos de facturación convencionales se entiende a aquellos que establecen un determinado precio por cada uno de los componentes software necesarios y por las unidades horarias o diarias de los recursos humanos necesarios, obteniendo el precio total como la suma de cada una de las partes implicadas. Pero también es posible establecer otros criterios de facturación basados en otros factores más orientados a los resultados obtenidos, como por ejemplo: el retorno de la inversión, el volumen de incremento en el negocio e la empresa, el nivel de ahorro conseguido, etc.

De acuerdo con esto, y a modo de ejemplo se presentan a continuación algunas ofertas reales de proyectos de venta de software y de consultoría:

1. Caso inicial: Una estimación de precio del software en base a los costes de desarrollo y explotación unitarios del software: 150-200.000 Euros mínimo
2. *Porting*: Proyecto para migrar un sistema basado en COBOL de un dialecto de una máquina específica a otra máquina: 1,5 Millones de Euros
3. Análisis del sistema: Proyecto para identificar malos hábitos de desarrollo y algoritmos poco eficientes: Porcentaje del ahorro de costes de máquina conseguido durante el primer año. Aproximadamente 500.000 Euros
4. Optimización de código: Proyecto de análisis automático del código existente, identificando un conjunto discreto de sentencias y usos no óptimos y sustituyéndolos por las sentencias adecuadas: 1.000.000 Euros
5. Establecimiento de un sistema de control de calidad: Primera fase aplicada a un único departamento, 500.000 Euros

# Conceptos

En este capítulo se describen brevemente un conjunto de conceptos que se utilizan en el sistema

## COBOL

XXXXXXXXXXXXXXX

### Estructura

Un programa fuente COBOL está estructurado en divisiones que agrupan un conjunto de instrucciones del lenguaje asociadas a un propósito común. No todas las divisiones son obligatorias pero si existen deben indicarse de acuerdo con el siguiente orden:

IDENTIFICATION DIVISION.

PROGRAM-ID. Nombre-del-programa.

otras-sentencias-de-identification

[ENVIRONMENT DIVISION.

contenido-environment-division]

[DATA DIVISION.

contenido-data-division]

PROCEDURE DIVISION [USING …].

codigo-ejecutable

[END-PROGRAM]

Código 1: Estructura COBOL

* IDENTIFICATION DIVISION

Contiene información básica relativa al programa, la única clausula obligatoria es PROGRAM-ID que identifica el nombre del programa que será el utilizado para su invocación.

Se considera una buena práctica que este nombre coincida con el nombre del archivo fuente del mismo; sin embargo, en determinados sistemas operativos, como por ejemplo zOS de IBM o VOS de Stratus, se limita el nombre de los archivos a 8 caracteres

* ENVIRONMENT DIVISION

Contiene la información relativa al entorno en el cual el programa será compilado y ejecutado: memoria, tipo de ordenador, repositorio de funciones externas, etc. En esta división se deben declarar la lista y características de todos los ficheros a los que el programa puede acceder.

* DATA DIVISION

Contiene la declaración de todas las variables que pueden ser utilizadas por el programa.

* PROCEDURE DIVISION

Contiene el código del programa.

### Variables

En un programa COBOL todas las variables deben haber sido declaradas en DATA DIVISION y son accesibles a todo el programa.

Una de las principales características que diferencia a este lenguaje de otros lenguajes de programación estructurada es que las variables se agrupan en números de nivel formando estructuras de tipo registro que, a su vez, pueden ser “*redefinidas*” formando estructuras unión.

Desde el punto de vista de los números de nivel, se aplican las siguientes restricciones:

* Nivel 77: Define una variable que está dividida en otras variables.
* Nivel 01: Define una variable o un conjunto de variables.
* Niveles 02-49: Definen variables o conjuntos de ellas que están incluidas en un nivel superior, siendo el primer nivel obligatoriamente el nivel 01.

Cuando se referencia una variable, el valor de esta comprende todas las variables que dependen de ella. Por ejemplo, la ejecución del siguiente programa:

\* SDP DESCRIPTION

\* DOC - Demuestra el uso de los niveles en la definicion de

\*> las variables

\* SDP END

IDENTIFICATION DIVISION.

PROGRAM-ID. DOCB0001.

DATA DIVISION.

WORKING-STORAGE SECTION.

77 VAR77 PIC X(10) VALUE 'VAR77'.

01 VAR01.

03 ABC PIC X(03) VALUE 'ABC'.

03 NIVEL03.

05 LETRAD PIC X(01) VALUE 'D'.

05 LETRAE PIC X(01) VALUE 'E'.

05 LETRAF PIC X(01) VALUE 'F'.

PROCEDURE DIVISION.

PERFORM DISPLAY-VARIABLES.

MOVE SPACES TO NIVEL03.

DISPLAY 'TRAS INICIAR NIVEL03 A ESPACIOS'.

PERFORM DISPLAY-VARIABLES.

STOP RUN.

DISPLAY-VARIABLES.

DISPLAY 'VARIABLE VAR77 ES : ' VAR77.

DISPLAY 'VARIABLE VAR01 ES : ' VAR01.

DISPLAY 'VARIABLE NIVEL03 ES : ' NIVEL03.

DISPLAY 'VARIABLE LETRAE ES : ' LETRAE.

Código 1: DOCB0001 - Declaración de variables

Produce el siguiente resultado:

$ ./DOCB0001.exe

VARIABLE VAR77 ES : VAR77

VARIABLE VAR01 ES : ABCDEF

VARIABLE NIVEL03 ES : DEF

VARIABLE LETRAE ES : E

TRAS INICIAR NIVEL03 A ESPACIOS

VARIABLE VAR77 ES : VAR77

VARIABLE VAR01 ES : ABC

VARIABLE NIVEL03 ES :

VARIABLE LETRAE ES :

### Flujo de ejecución

Salvo que se utilice la cláusula ENTRY el flujo del programa empieza en la primera sentencia que aparezca después de PROCEDURE DIVISION y es ejecutado secuencialmente hasta que se produce una de las siguientes situaciones:

1. Encuentra el final de fichero
2. Encuentra END PROGRAM, indicando de manera explícita el fin del fichero fuente
3. Encuentra STOP RUN, indicando el final de la ejecución del proceso
4. Encuentra GOBACK, indicando el final de la ejecución del módulo actual

Esto puede dar lugar a comportamientos que los programadores en otros lenguajes de programación puedan considerar “extraños”:

Por ejemplo, el siguiente código produce el mismo resultado:

|  |  |  |
| --- | --- | --- |
| PROCEDURE DIVISION.  DISPLAY 'Hello World'. |  | PROCEDURE DIVISION.  INICIO.  DISPLAY 'Hello World'. |
|  |  |  |
| **$ ./DOCB0101.exe**  **Hello World** |  | **$ ./DOCB0102.exe**  **Hello World** |
| Código 2: DOCB0101 |  | Código 3: DOCB0102 |

Pero el siguiente código genera un resultado diferente del esperado:

|  |
| --- |
| PROCEDURE DIVISION.  PERFORM INICIO.  INICIO.  DISPLAY 'Hello World'. |
|  |
| **$ ./DOCB0201.exe**  **Hello World**  **Hello World** |
| Código 4: DOCB0201 |

### Rutinas

El lenguaje COBOL está fuertemente orientado a la modularización y la reutilización del código en base al uso de rutinas o módulos.

Una rutina es un programa COBOL normal que es invocado por otro programa y puede o no recibir parámetros por referencia (opción por defecto) o por valor.

Existen dos modos de invocación a una rutina:

* Llamada estática: CALL “nombre\_rutina” [USING parámetros]
* Llamada dinámica: CALL variable [USING parámetros]

El uso de uno u otro modo afecta directamente al modo de funcionamiento del programa, cuando se utilizan llamadas estáticas a módulos:

* En el momento de compilación se deben indicar todos los módulos utilizados
* Se genera un único ejecutable con todos los módulos incluidos en el mismo. Por esto es necesario declararlos todos en el momento de compilación, ya que en caso contrario, se produce un error en tiempo de ejecución
* Cualquier cambio posterior en las rutinas incluidas no se ven reflejadas en el programa principal.

Cuando se utiliza una llamada dinámica:

* La rutina no es incluida en el ejecutable y se carga en tiempo de ejecución como si fuera una librería dinámica
* La rutina es compilada de manera independiente por lo que siempre se utiliza la última versión de la misma

En cualquiera de los dos modos de invocación el módulo invocado es cargado en el espacio de direcciones del módulo principal **una única vez**; es decir, si en alguno de los módulos implicados en una unidad de ejecución ya está cargada una determinada rutina, esta no se vuelve a cargar y su estado interno se mantiene a lo largo de toda la ejecución.

Para aclarar este concepto consideremos el siguiente árbol de módulos, donde las flechas continuas indican llamadas estáticas, las discontinuas llamadas dinámicas, y los números el orden de ejecución:



Diagrama 1: Árbol llamadas estáticas

Cada programa, cuando es invocado, incrementa en uno un contador y muestra su valor. El resultado es:

$ ./DOCB0301.exe

EJECUTA DOCB0311 DESDE DOCB0301 CON CONTADOR = 001

EJECUTA DOCB0321 DESDE DOCB0311 CON CONTADOR = 001

EJECUTA DOCB0322 DESDE DOCB0311 CON CONTADOR = 001

EJECUTA DOCB0321 DESDE DOCB0322 CON CONTADOR = 002

EJECUTA DOCB0321 DESDE DOCB0301 CON CONTADOR = 003

## Complejidad ciclomática

La complejidad ciclomática según fue definida por Thomas McCabe en su célebre articulo A Complexity Measure [CAB1] establece un método para medir y controlar el número de caminos diferentes que puede seguir el flujo de un programa en base a la construcción de grafos.

Este método, por un lado, identifica el conjunto de pruebas mínimo que sería necesario realizar para garantizar que se han ejecutado al menos una vez todas las sentencias de código de ese programa, y por otro, establece una métrica del nivel de complejidad y en consecuencia, de mantenibilidad, del código.

Considerando que un programa ejecuta sus instrucciones secuencialmente, la única forma de que se establezca un cambio en el flujo del programa es a través de las denominadas “sentencias de control de flujo”. En COBOL las únicas sentencias de control de flujo disponibles son:

* Sentencia IF: Ejecuta el conjunto de sentencias si se cumple la condición
* Sentencia EVALUATE: Es una forma compacta de agrupar un conjunto de sentencias IF dependientes de una variable
* Clausula UNTIL en sentencias PERFORM: La sentencia bifurca el control del programa mientras se cumplan las condiciones establecidas en la cláusula UNTIL

Cada una de estas situaciones se puede modelar como un grafo de acuerdo con los siguientes:

|  |
| --- |
|  |
|  |

Diagrama 2: Grafos de control de flujo

Nótese que los predicados complejos, como por ejemplo IF A AND B THEN X se han expandido de manera que cada condición de bifurcación genera un predicado simple. Esto es debido a que la complejidad ciclomática se establece como:

Lo cual plantea el problema de construir el grafo y analizar el número de aristas y el número de nodos existentes en él; sin embargo, si conocemos que los nodos están todos formados por predicados simples; es decir, no contienen operadores lógicos AND u OR, entonces la complejidad ciclomática también se puede expresar como:

Con lo que no es estrictamente necesario construir el grafo para obtener a complejidad ciclomática de un determinado bloque de software. Basta con identificar el número de sentencias de control de flujo existentes y el número de operadores lógicos presentes para obtener la complejidad ciclomática.

Por ejemplo, consideremos el siguiente código:

|  |
| --- |
| PROCEDURE DIVISION.  1 PERFORM PROCESO1  2=3+4 IF A = 1 AND B > 2 THEN  5 PERFORM PROCESO2  END-IF  6 PERFORM PROCESO3  STOP RUN. |

Se pueden dibujar dos grafos:

|  |  |
| --- | --- |
|  |  |
| Grafo condensado | Grafo expandido |

Diagrama 3: Grafo condensado vs grafo expandido

En el segundo caso, nos basta con identificar la sentencia IF y el operador AND para identificar que existen 2 nodos con predicados simples, con lo que la complejidad ciclomática es 1 + 2 = 3. Que se corresponde con los casos de prueba:

1. A y B
2. ⌐A y B
3. A y ⌐B

Nótese que el caso ⌐A y ⌐B que sería la otra opción posible para efectuar una cobertura total, recorrería caminos del grafo ya recorridos por los casos anteriores.

Por último, la complejidad ciclomática de la concatenación de cualquiera de las estructuras básicas presentadas en la suma de todas ellas:

|  |  |
| --- | --- |
|  | Predicados simples: 2  Complejidad ciclomática: 1 + 2 = 3 |

Diagrama 4: Grafo con varios bloques

## Tiempos y Consumos

Cuando un proceso se ejecuta, el usuario final percibe el tiempo como la fracción de tiempo que transcurre desde que solicita una información hasta que la recibe, y esa es la percepción que tiene acerca de la efectividad del programa (aparte de la adecuación de la información recibida a lo realmente esperado por el usuario final).

Sin embargo, ese tiempo percibido es la suma de todos los tiempos invertidos en todos los componentes de la petición, lo cual puede dar lugar a obtener conclusiones erróneas acerca del comportamiento del proceso. Por ejemplo, supongamos que un usuario solicita a través una red un cierto dato, y que esa red, por congestión o por otros motivos ofrece un mal rendimiento y la respuesta le tarda en llegar 4,01 segundos.

El usuario final, con esa información, puede concluir que el programa no da el rendimiento adecuado y dejar de usarlo; sin embargo si consideramos cada uno de las tareas realizadas y los componentes implicados podemos obtener otras conclusiones:

|  |  |  |
| --- | --- | --- |
|  | Acción | Tiempo |
| 1 | El usuario realiza la petición | 0 sg. |
| 2 | La petición se envía por la red al servidor | 2 sg. |
| 3 | El programa obtiene la información y la devuelve | 0,01 sg. |
| 4 | La información se devuelve por la red al usuario | 2 sg. |
| 5 | El usuario recibe la información | 0 sg. |
|  | **Total** | **4,01 sg.** |

Realmente el tiempo que ha invertido el proceso en obtener la información ha sido de 0,01 segundos, con lo que a priori no se podría decir que el rendimiento del proceso no ha sido el adecuado, pero sí identificar que hay un problema de red.

Al tiempo que transcurre desde que se inicia una acción hasta que obtenemos una respuesta lo llamaremos ***Elapsed***.

En este ejemplo, el tiempo Elapsed hace referencia al tiempo que percibe el usuario, y es de 4,01 segundos. Pero si el punto de medida lo tomamos en otro punto, por ejemplo, a la entrada del proceso, entonces nuestro tiempo Elapsed sería de 0,01 segundos.

Ahora bien, este nuevo tiempo Elapsed también está formado por la suma de otros tiempos parciales, más concretamente, en el caso de un proceso ejecutable este tiempo es la suma de los siguientes tiempos:

* **Tiempo de CPU de usuario**: Tiempo que el proceso está ejecutando código
* **Tiempo de CPU de *Kernel***: Tiempo que el sistema está realizando procesos en nombre del proceso
* **Tiempo de *Suspend***: Tiempo que el proceso puede estar detenido por decisión del sistema; por ejemplo, en un sistema que emula multitarea, en un instante dado solo hay un proceso ejecutándose y el resto de procesos están parados
* **Tiempo de *Idle***: Tiempo que el proceso está esperando sin realizar ninguna acción, por ejemplo, porque está esperando una entrada del usuario
* **Tiempo de *Wait***: Tiempo que el proceso no está realizando ninguna acción porque está esperando que se le asigne algún recurso
* **Tiempo de *Subsistemas***: En algunos Sistemas Operativos también se tiene en cuenta el tiempo consumido por ciertos subsistemas en nombre del proceso, de manera que estos consumos son imputados al proceso que lo solicito en lugar de al subsistema. Por ejemplo, cuando solicita una consulta a un SGDB, el proceso lo realiza el propio SGDB, por lo que sería un tiempo de Wait (Estoy esperando por los datos), pero en estos Sistemas Operativos, este consumo aparece contabilizado como tiempo del subsistema, de manera que si se miden los consumos del SGDB únicamente ofrece los realizados por las tareas de gestión y administración.

Para las métricas implementadas en la versión actual del sistema, no es relevante conocer si el tiempo de CPU se ha consumido en modo Usuario o en modo Kernel ni tampoco cual ha sido el motivo concreto por el que el proceso ha estado detenido, por lo que consideramos los siguiente tiempos:

**Tiempo *Elapsed* (TE)**: Tiempo total transcurrido entre el inicio y el fin de una tarea determinada.

**Tiempo CPU (TC)**: Tiempo de CPU consumida en cualquiera de los modos de ejecución por esa tarea.

**Tiempo *Suspend* (TS)**: La diferencia entre esos dos tiempos; es decir, el tiempo en el que la tarea no ha ejecutado ninguna instrucción de la CPU.

En un escenario ideal, el tiempo Elapsed debería ser prácticamente igual al tiempo de CPU, lo cual indicaría que todos los recursos estan disponibles en el momento adecuado y que no ha habido ninguna colisión con otros procesos.

En un escenario real, siempre existe un cierto nivel de tiempo Suspend, pero un nivel alto, en términos porcentuales de este tiempo, indica que:

* El proceso no está funcionando adecuadamente aunque posiblemente no sea culpa de un mal diseño del programa
* Existe algún problema en alguno o en varios de los componentes y recursos externos que necesita el proceso para su ejecución.

# Visión general

## Arquitectura

# Parser

# Trapper

Este componente es el encargado:

1. Obtener los tiempos y consumos del sistema imputándoselos al componente correspondiente
2. Generar los mensajes adecuados con esa información
3. Enviar los mensajes generados al Collector

## Consideraciones de diseño

A la hora de implementar este componente hay que tener en cuenta las siguientes consideraciones:

### Overhead (Sobrecarga)

Dentro del flujo de ejecución del conjunto de programas asociados a una sesión cada componente monitorizado: Módulos, párrafos, sentencias CALL, … va a invocar a este componente, en principio, dos veces: a la entrada y a la salida de ese componente.

Esto implica que, respecto al módulo original, al módulo monitorizado se le incorporan otros bloques de proceso, que no son relevantes desde el punto de vista funcional pero significativos desde el punto de vista de la sobrecarga que la ejecución de estos bloques supone respecto a los tiempos de ejecución y los consumos de CPU.

Por ello, el código necesario para realizar estos controles tiene que ser lo mas eficiente posible en el sentido de requerir el menos número de instrucciones posibles y el menor consumo de recursos posibles.

Si descartamos el lenguaje ensamblador por su dependencia del procesador sobre el que se ejecuta el programa, lo cual limitaría la portabilidad del sistema y su dificultad a la hora de mantenerlo; la siguiente elección es desarrollar este componente en C.

### Flujo de ejecución

Una de las características del lenguaje COBOL es que este no permite la creación de hilos de ejecución por lo que el conjunto de programas asociados a una sesión se ejecutan siempre en un único procesador[[1]](#footnote-2) y sus sentencias se ejecutan secuencialmente

Esto implica que la ejecución normal de cualquier programa COBOL, y por extensión cualquier programa *monohilo*, se comporta como la ejecución de un conjunto de bloques de código de manera secuencial que pueden ser monitorizados, salvo que se produzca un salto incondicional en el flujo de ejecución, con una pila de tiempos en las que se pone en la cima el instante de inicio del proceso de un determinado bloque y cuando este bloque finaliza, se extrae de la pila el instante de inicio, obteniendo el consumo por una simple diferencia de tiempos.

Por ejemplo, dado un programa similar al siguiente:

000009 PROCEDURE DIVISION.

000010 INICIO.

000011 *OTRAS SENTENCIAS …*

000012 PERFORM PROCESO1.

000013 PERFORM PROCESO2.

000014 *OTRAS SENTENCIAS …*

000015 STOP RUN.

000017 PROCESO1.

000018 *OTRAS SENTENCIAS ….*

000019 PERFORM PROCESO11.

000020

000021 PROCESO11.

000022 *OTRAS SENTENCIAS …*

000023

000024 PROCESO2.

000025 *OTRAS SENTENCIAS …*

000026

Código 5: Captura de tiempos

Los puntos a monitorizar serian:

|  |  |
| --- | --- |
| 000010 | Inicio programa (Tiempo relativo = 0) |
| 000011 | Inicio párrafo INICIO |
| 000015 | Fin párrafo INICIO  Fin programa |
| 000017 | Inicio párrafo PROCESO1 |
| 000019 | Fin párrafo PROCESO1 |
| 000022 | Inicio PROCESO11 |
| 000023 | Fin PROCESO11 |
| 000025 | Inicio PROCESO2 |
| 000026 | Fin PROCESO3 |

Y considerando el flujo de ejecución mostrado y asumiendo que cada bloque consume una unidad de tiempo la pila de tiempos tendría un aspecto similar al mostrado en la siguiente figura:



Diagrama 5: Ejemplo de pila de monitorización

Lo cual da los siguientes valores de consumos de tiempos:

| Tiempo | Acción | Timer | Consumo |
| --- | --- | --- | --- |
| 0 | Push Proceso | 0 |  |
|  | Push INICIO | 0 |  |
| 1 | Push PROCESO1 | 1 |  |
| 2 | Push PROCESO11 | 2 |  |
| 3 | Pop PROCESO11 | 3 | 3 – 2 = 1 |
| 4 | Pop PROCESO1 | 4 | 4 – 2 = 2 |
|  | Push PROCESO2 | 4 |  |
| 5 | Pop PROCESO2 | 5 | 5 – 4 = 1 |
| 6 | Push Inicio | 6 | 6 – 0 = 6 |
|  | Push Proceso | 6 | 6 – 0 = 6 |

### Generación óptima de traps

Otro aspecto relevante a considerar a la hora de la implementación de este componente, directamente relacionado con la sobrecarga del proceso inherente a la captura de tiempos y generación de mensajes es la optimización de los mensajes que se deben enviar al *Collector* de manera que se mantenga en todo momento una información fiable y suficientemente detallada para poder extraer conclusiones de las mismas.

De acuerdo con la estructura de pila descrita anteriormente, una primera aproximación sería generar un mensaje cada vez que se saca un elemento de la pila (Este sería el modo de proceso definido como **DETAILED**)

Sin embargo, si observamos el siguiente código, utilizado como ejemplo para efectuar una ordenación de datos por el método de la burbuja:

PROCEDURE DIVISION.

INICIO.

CALL CARGA USING DATA-SORT.

PERFORM ORDENAR.

CALL DESCARGA USING DATA-SORT.

STOP RUN.

ORDENAR.

COMPUTE MAX = ITEMS

PERFORM BURBUJA VARYING MAX FROM MAX BY -1 UNTIL MAX = 1.

BURBUJA.

MOVE 1 TO IDX

PERFORM COMPARAR VARYING IDX FROM 1 BY 1 UNTIL IDX = MAX.

COMPARAR.

IF ORDEN(IDX) > ORDEN (IDX + 1)

MOVE ORDEN(IDX + 1) TO AUX

MOVE ORDEN(IDX) TO ORDEN(IDX + 1)

MOVE AUX TO ORDEN(IDX)

END-IF.

Código 6: Ordenación método de la burbuja

Suponiendo un conjunto de entradas de *n* = 1.000 elementos, observamos que al párrafo BURBUJA se ejecutará *n – 1* veces; es decir, *k = n – 1 = 999* y este, a su vez, invocará al párrafo COMPARAR *k – 1* veces cada vez; es decir, la primera llamadas a BURBUJA invocará a COMPARAR 999 veces, la segunda 998 veces, la tercera 997, etc. Hasta que se esté comparando el último elemento, lo cual, para el ejemplo presentado supone que COMPARAR se ejecutará de acuerdo con una progresión aritmética de 1 a 999 elementos con una diferencia de 1.

Parece lógico entonces considerar que la generación de más de 500.000 mensajes para un simple proceso de ordenación no es óptimo ni necesario (salvo en casos especiales) y que, sin pérdida de información se puede generar, al final de la ejecución del módulo, un mensaje por cada bloque de código monitorizado, que contenga los valores acumulados y la cuenta de las veces que se ha ejecutado.

Este modo de proceso es el que se define como **MODULE**.

Extendiendo la lógica descrita a las rutinas utilizadas; supongamos que la rutina CARGA, encargada de cargar los datos en memoria, en lugar de ser invocada una única vez se ejecuta de acuerdo con el siguiente pseudocódigo[[2]](#footnote-3):

CALL CARGA USING ABRIR-FICHERO

MIENTRAS HAYA-REGISTROS

CALL CARGA USING LEER-REGISTRO

ADD REGISTRO TO TABLA

FIN MIENTRAS

CALL CARGA USING CERRAR-FICHERO

Se produce la misma explosión de generación de mensajes y la consiguiente pérdida de rendimiento. Con lo cual se puede extender la filosofía descrita anteriormente y esperar al final de la sesión para enviar los acumulados tanto de los bloques como de los módulos ejecutados.

Este modo de proceso es el que se define como **GLOBAL**.

A modo de resumen, para el supuesto planteado de un proceso de ordenación de 1.000 elementos donde la carga de datos se realiza en otro módulo que es invocado una vez para recuperar cada registro se muestra a continuación el volumen de mensajes generados para cada una de las opciones:

| Mensaje | DETAILED | MODULE | GLOBAL |
| --- | --- | --- | --- |
| Sesion | 2 | 2 | 2 |
| Modulo | 1.004 | 1.004 | 3 |
| Call | 1.003 | 2 | 2 |
| Parrafos | 502.005 | 1.007 | 8 |
| **Total** | **504.014** | **1.015** | **15** |

### Generación óptima de mensajes

## Implementación

La librería es llamada, por cada componente monitorizado dos veces: Una cuando entra en el componente y otra cuando sale.

Para realizar el proceso, la librería se apoya en tres estructuras de datos:

* Una pila donde se almacenan los tiempos de inicio de cada componente
* Una pila donde se almacena los módulos activos en el árbol de llamadas
* Una lista donde se almacena la información de cada modulo



Diagrama 6: Relación de las estructuras de SDPTRAPB

Y dos grupos de mensajes:

* Mensajes de inicio (bit menos significativo a 0)
* Mensajes de fin (bit menos significativo a 1)

### Diagrama de flujo

El diagrama de flujo de alto nivel del proceso



Diagrama 7: Flujo principal SDPTRAPB



Diagrama 8: Flujo de mensaje de fin

## Arquitectura



Diagrama 9: Arquitectura SDPTRAPB

### Ficheros fuente

| Modulo | Descripción |
| --- | --- |
|  |  |
| SDPTRAPB | Punto de entrada a la librería.  Se encarga de gestionar el flujo del proceso |
|  |  |
| amqsdp | Gestiona el envío de mensajes |
|  |  |
| cob2c | Convierte la COPY COBOL a una estructura SDP y viceversa |
|  |  |
| md5 | Genera una firma digital MD5 |
|  |  |
| sha256 | Genera una firma digital SHA-256 en función del procesador |
|  |  |
| sha256be | Genera una firma digital SHA-256 en arquitecturas BigEndian |
|  |  |
| sha256le | Genera una firma digital SHA-256 en arquitecturas LittleEndian |
|  |  |
| messages | Gestiona el almacenamiento de los Timers y la generación de mensajes |
|  |  |
| modulos | Gestiona la lista de módulos |
|  |  |
| pila | Gestiona la creación de pilas |
|  |  |
| salw32 | Abstrae las peculiaridades del Sistema Windows |
|  |  |
| timerWin | Obtiene los consumos del sistema en sistemas Windows |
|  |  |
| timerLnx | Obtiene los consumos del sistema en sistemas Linux |
|  |  |

Tabla 1: Listado de módulos fuente SDPTRAPB

### Ficheros de cabecera

De manera general, los ficheros de cabecera cuyo nombre coincide con un fichero de código, definen las funciones públicas de ese módulo.

El resto de ficheros de cabecera se describen a continuación:

| Modulo | Descripción |
| --- | --- |
|  |  |
| cmqc.h | Fichero de cabecera del sistema de mensajería MQ Series |
|  |  |
| global.h | Define las variables de uso global dentro del sistema. |
|  |  |
| jggsal.h | Abstrae el sistema de las características específicas de un Sistema Operativo concreto.  Redefine funciones de uso común e incluye las librerías especificas de ese Sistema Operativo |
|  |  |
| sdpConfig.h | Establece las constantes y macros que definen el comportamiento global del sistema |
|  |  |
| sdpTypes.h | Define las estructuras de datos usadas en la librería |
|  |  |
| timer.h | Incluye el módulo de cálculo de Timers adecuado para la plataforma |
|  |  |
| trapCodes.h | Define los códigos de los mensajes.  Estos valores se deben corresponder con sus equivalentes en los otros componentes del sistema |

Tabla 2: Listado de ficheros de cabecera SDPTRAPB

### Dependencias externas

### Estructuras de datos

## Mensajes

### Inicio de sesión

Este mensaje se envía en el momento que se genera independientemente del estado del buffer de mensajes.

Marca el inicio de una sesión.

|  |  |  |
| --- | --- | --- |
| Orden | Campo | Descripción |
| 1 | Tipo | Identificador del mensaje. Valor: **0** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión  Valor: **0** |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Principal | Firma digital del módulo principal |
| 5 | Usuario | Identificador del usuario |
| 6 | Timestamp | Timestamp del sistema |
| 7 | Elapsed | Tiempo consumido desde que se invocó el programa |
| 8 | CPU | Tiempo de CPU consumido desde que se invocó el programa |

Tabla 3: Formato mensaje BEG\_SESION

### Fin de sesión

Este mensaje se envía cuando finaliza el módulo principal; es decir, cuando se han vaciado las pilas.

En caso de que el programa acabe de manera inesperada, este mensaje no se enviará y se detectará el fallo del programa

| Orden | Campo | Descripción |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **1** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Principal | Firma digital del módulo principal |
| 5 | Usuario | Identificador del usuario |
| 6 | Timestamp | Timestamp del sistema |
| 7 | Elapsed | Tiempo consumido desde que se invocó el programa |
| 8 | CPU | Tiempo de CPU consumido desde que se invocó el programa |

Tabla 4: Formato mensaje END\_SESION

### Módulo

Este mensaje se envía cuando finaliza la ejecución de un módulo

| Orden | Campo | Descripción |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **2** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Etiqueta | Nombre del módulo |
| 6 | Veces | Número de veces que se ha ejecutado |
| 7 | Elapsed | Tiempo total consumido |
| 8 | CPU | Tiempo total de CPU consumido |
| 9 | IntElapsed | Tiempo total consumido en el módulo |
| 10 | IntCPU | Tiempo total de CPU consumido en el módulo |

Tabla 5: : Formato mensaje BEG\_MODULE

### Párrafo

Este mensaje se envía cuando finaliza la ejecución de un párrafo

| Orden | Campo | Descripción |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **4** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Etiqueta | Nombre del párrafo |
| 6 | Veces | Número de veces que se ha ejecutado. Valor: 1 |
| 7 | Elapsed | Tiempo total consumido |
| 8 | CPU | Tiempo total de CPU consumido |
| 9 | IntElapsed | Tiempo total consumido en el párrafo |
| 10 | IntCPU | Tiempo total de CPU consumido en el párrafo |

Tabla 6: Formato mensaje BEG\_PARAGRAPH

### Call

Este mensaje se envía cuando vuelve de una sentencia CALL

| Orden | Campo | Descripción |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **8** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Calling | Firma digital del módulo llamante |
| 5 | Called | Firma digital del módulo llamado |
| 6 | Etiqueta | Nombre del modulo llamado |
| 7 | Veces | Número de veces que se ha ejecutado |
| 8 | Elapsed | Tiempo total consumido |
| 9 | CPU | Tiempo total de CPU consumido |
| 10 | IntElapsed | Tiempo total consumido en el módulo |
| 11 | IntCPU | Tiempo total de CPU consumido en el módulo |

Tabla 7: Formato mensaje BEG\_CALL

### Cobertura

Este mensaje se envía cuando una vez por módulo. Contiene los flags de los bloques de código que se han ejecutado

| Orden | Campo | Descripción |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **256** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Flags | Secuencia de ceros y unos indicando que bloques se han ejecutado |

Tabla 8: Formato mensaje MSG\_COVER

### Ficheros

Este mensaje se envía cuando una vez por módulo. Contiene los contadores de los diferentes accesos que se han hecho a los posibles ficheros utilizados en ese modulo.

Cada fichero contempla 8 tipos de accesos (Apertura, Cierre, Lectura, Inserción, Actualización, Borrado, Lectura secuencial, Modificación)

| Orden | Campo | Descripción |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **512** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Accesos | Secuencia con los contadores de los diferentes accesos realizados sobre cada uno de los ficheros separados por el carácter ‘X’ |

Tabla 9: : Formato mensaje MSG\_PERSISTENCE

### Perform

Este mensaje se envía cuando una vez por módulo. En lugar de controlar el inicio y fin de un párrafo, se controla el inicio y el fin de la llamada a ese párrafo

| Orden | Campo | Descripción |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **1024** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Llamadas | Secuencia con los contadores del uso realizado de cada párrafo separados por el carácter ‘X’.  La identificación del párrafo se determina por su posición con respecto a la declaración de los párrafos identificados en el momento de compilación.  Ejemplo: El valor contenido antes de la primera ‘X’ (Primer párrafo) se corresponde con el párrafo identificado como 1 en la tabla de declaración de párrafos de esa versión de ese módulo |

Tabla 10: Formato mensaje MSG\_PERFORM

# Collector

Each DIVISION may consist of a variety of SECTIONs and each SECTION consists of one or more PARAGRAPHs. A PARARAPH consists of SENTENCEs, each of which consists of one or more STATEMENTs.

Desde el punto de vista del código, este se puede ver formado por:

Secciones : Una sección esta formada por uo o mas párrafos

Parrafo: Un párrafo esta formado uno o mas sentencias

Sentencia: UN conjunto de bloques

Bloque: Un conjunto de instrucciones que se ejecutarán como una unidad; en el sentido de que, si el flujo de programa alcanza ese punto, se ejecutarán todas ellas

Instrucción: Cada uno de los verbos del lenguaje

## Insercion de variables

Las variables necesarias para el fucnionamiento se alamacenan en Working

Puede no haber esa sección, entonces se mira si hay data división, si la hay hay otras cosas para ponerla en su sitio

La secuencia es local-storage, linkage, report, screen si no hay ninguna de ellas

Va encima de procedure

Si no hay data división

Va antes de procedure incluyendo data division

### Collector / Parser

### Collector / Trapper

### Collector / Consolidator

# Modelo de datos

En este capítulo se presenta el modelo de datos subyacente del sistema y la relación entre sus tablas.

Con el objeto de no extender demasiado este capítulo, se realiza una breve descripción de las tablas que conforman el modelo, detallando únicamente aquellos aspectos que puedan ser relevantes o específicos de este modelo con respecto a lo que sería un modelo de datos clásico diseñando en tercera forma normal.

La definición completa del modelo de datos, así como sus valores por defecto se pueden consultar en el Anexo: XXXXXXX

## Visión general

El modelo de datos sobre el que se apoya el sistema está estructurado en función del componente del sistema que lo alimenta y el bloque funcional que representa y agrupadas de acuerdo con el siguiente criterio de nomenclatura:

| Prefijo | Descripción |
| --- | --- |
| CFG | Tablas de configuración del sistema y mensajes |
| LOG | Tablas de *logging* |
| MOD | Tablas con la información estática de los módulos |
| SDP | Tablas base con la información de áreas, aplicaciones y módulos |
| SES | Tablas con la información relativa a las sesiones |
| SUM | Tablas con la información dinámica de los módulos |
| TRP | Tablas con los mensajes recibidos durante las sesiones de ejecución |

Tabla 11: Prefijos de tablas

Las relaciones que se establecen entre los diferentes grupos de tablas se muestran en el siguiente diagrama



Diagrama 10: Relación entre grupos de tablas

* Una aplicación puede tener cero o varias aplicaciones, en este último caso la aplicación se considerará un área.
* Dado que las aplicaciones las define un administrador, una aplicación puede tener cero o varios módulos.
* Puesto que un módulo se da de alta en el sistema cuando se analiza, un módulo siempre tendrá al menos una versión de su código fuente.
* Una versión de un módulo tendrá:
  + Su información estática almacenada en el conjunto asociado de tablas de información de ese módulo (Tablas MOD\_\*).
  + Su información dinámica almacenada en el conjunto asociada de tablas sumarizadas de ejecución (Tablas SUM\_\*)
* La versión de un módulo no tendrá información de ejecución asociada hasta que no se ejecute al menos una vez.
* Por cada sesión de ejecución (Tablas TRP\_\*), se generará la información consolidada en las tablas de resumen de la sesión (Tablas SES\_\*)

## Tablas generales

Son las tablas que están asociadas al sistema en general, sin relación con un determinado módulo.

|  |  |
| --- | --- |
| Tabla | Descripción |
|  |  |
| CFG\_CONFIGURACION | Contiene la configuración global del sistema.  Esta tabla se corresponde, cuando el modelo de datos no está disponible, con un fichero de propiedades |
|  |  |
| CFG\_CODIGOS | Contiene la descripción de los códigos numéricos utilizados en el sistema agrupados por los idiomas disponibles |
|  |  |
| MOD\_FUENTE\_ERR | Se trata de un caso especial.  Contiene el código fuente de aquellos programas que no han podido ser analizados correctamente o que han generado algún error al compilar el nuevo código fuente generado por el Parser.  Su uso está destinado exclusivamente a los propietarios del sistema para poder identificar posibles errores en el sistema o incorporar nuevas funcionalidades |

Tabla 12: Tablas generales

### CFG\_CONFIGURACION

Como se ha indicado, esta tabla contiene la configuración global del sistema y se corresponde con un fichero de propiedades que utilice el formato clave=valor.

Esta tabla se corresponde, cuando el modelo de datos no está disponible, con un fichero de propiedades.

|  |  |  |  |
| --- | --- | --- | --- |
| Campo | Tipo | Long. | Descripción |
|  |  |  |  |
| Clave | VCHAR | 127 | Nombre simbólico del parámetro de configuración |
|  |  |  |  |
| Valor | VCHAR | 255 | Valor actual del parámetro de configuración |
|  |  |  |  |
| Grupo | INT |  | Indicador que permite agrupar los parámetros de configuración por su aplicación funcional. |
|  |  |  |  |
| Tipo | INT |  | Indica el tipo de dato asociado al valor del parámetro: Número, Fecha, Directorio, etc. |
|  |  |  |  |
| Mask | VCHAR | 127 | Máscara de validación del parámetro |
|  |  |  |  |
| Minimo | VCHAR | 32 | Valor mínimo posible |
|  |  |  |  |
| Maximo | VCHAR | 32 | Valor máximo posible |
|  |  |  |  |
| Tooltip | INT |  | Identificador del mensaje descriptivo |

Tabla 13: Tabla CFG\_CONFIGURATION

El motivo por el que se han incorporado más campos a esta tabla aparte de los clásicos “clave” y “valor” es facilitar la administración y gestión de los parámetros mediante una interfaz web.

**Grupo**: Agrupa los parámetros por la funcionalidad que representan, lo cual permite presentarlos de manera automática en pestañas cuando el volumen de parámetros de configuración es excesivo y establecer criterios de autorización sobre quien puede modificar los parámetros.

**Tipo**: Dado que cada parámetro puede ser de un tipo de dato diferente: Números, cadenas, fechas, directorios, etc. El incluir un campo que defina el tipo de dato permite almacenar todos los valores en un mismo campo genérico de tipo texto sin perjuicio del valor contenido y realizar validaciones automáticas en el momento de su modificación.

**Mask**: Cuando está informado, establece una expresión regular que el valor del parámetro debe cumplir. Por ejemplo, el parámetro “parser.margin.left” especifica la posición de inicio del código fuente de acuerdo con los estándares COBOL y solo podría tomar dos valores: 0 cuando no se mantienen las áreas COBOL o 7 cuando sí. En consecuencia es un tipo numérico y su máscara es: [0|7]{1}. Un caso especial, no implementado en esta versión, es cuando el parámetro puede tomar únicamente un valor de un conjunto discreto de opciones; en este caso este campo contiene la clave asociada a la tabla que mantiene ese conjunto de valores, de manera que estos se presenten como lista de opciones.

**Mínimo**: Si está informado indica el valor mínimo que puede tomar el parámetro

**Máximo**: Si está informado indica el valor máximo que puede tomar el parámetro.

**Tooltip**: Establece el código del mensaje que se mostrará en la pantalla de administración Web.

## Tablas base

Contienen la información relativa a las áreas, aplicaciones y módulos activos en el sistema.

Está formado por las siguientes tablas:

|  |  |
| --- | --- |
| Tabla | Descripción |
|  |  |
| SDP\_APLICACIONES | Contiene la lista de aplicaciones activas en el sistema en forma de árbol mediante el atributo padre. |
|  |  |
| SDP\_MODULOS | Contiene la lista de módulos activos en el sistema |
|  |  |
| SDP\_REL\_APP\_MOD | Contiene las expresiones regulares que permiten, en el momento de dar de alta un módulo, asociarlo a una determinada aplicación |

Tabla 14: Tablas SDP

El diagrama de relación, considerando también la relación intrínseca entre un módulo y sus versiones se muestra a continuación:



Diagrama 11: E/R Grupo tablas SDP

### SDP\_APLICACION

Contiene la información relativa a un área o aplicación

|  |  |  |  |
| --- | --- | --- | --- |
| Campo | Tipo | Long. | Descripción |
|  |  |  |  |
| Aplicación | VCHAR | 32 | Nombre de la aplicación |
|  |  |  |  |
| Id | BIGINT |  | Identificador único de la aplicación |
|  |  |  |  |
| Descripción | VCHAR | 512 | Descripción de la aplicación |
|  |  |  |  |
| Padre | BIGINT |  | Identificador de la aplicación de la que depende.  0 – No depende de ninguna otra aplicación |
|  |  |  |  |
| Volumen | INT |  | Número de módulos de la aplicación |
|  |  |  |  |
| Uid | VCHAR | 32 | Usuario que realizó la última actualización |
|  |  |  |  |
| Tms | TMS |  | Timestamp de la última vez que la aplicación se actualizó |

Tabla 15:Tabla SDP\_APLICACION

Volumen:

Las aplicaciones son dadas de alta en el sistema por un administrador autorizado, pero los módulos son dados de alta automáticamente la primera vez que se analizan y dados de baja o decomisados por un administrador cuando ya no son utilizados o no se van a seguir monitorizando.

Esto implica que no se puede conocer el volumen de módulos que forman una aplicación por el número de módulos existentes en la tabla de Módulos, pero si es razonable asumir que todos los módulos de una aplicación han sido monitorizados al menos en un momento de su ciclo de vida.

Este campo se actualiza cada vez que se da un módulo de alta en el flujo automático del proceso y se vuelve a actualizar únicamente cuando un administrador da de baja un módulo.

En este contexto se establece una diferencia entre el hecho de indicar que un módulo ya no forma parte del sistema y que aunque sigue siendo operativo, no se desea monitorizar.

### SDP\_REL\_APP\_MOD

Contiene la información que permite asociar un módulo a una aplicación

|  |  |  |  |
| --- | --- | --- | --- |
| Campo | Tipo | Long. | Descripción |
|  |  |  |  |
| idAppl | BIGINT |  | Identificador de la aplicación |
|  |  |  |  |
| Mask | VCHAR | 12 | Expresión regular para asociar sus módulos |
|  |  |  |  |
| Fijo | INT |  | Número de caracteres no especiales |
|  |  |  |  |
| Peso | INT |  | Valor numérico relativo de la importancia de la mascara |
|  |  |  |  |
| Uid | VCHAR | 32 | Usuario que realizó la última actualización |

Tabla 16: Tabla SDP\_REL\_APP\_MOD

La asociación de un módulo a una determinada aplicación se realiza de forma automática en función del contenido de esta tabla.

Cada aplicación tiene al menos una máscara que indica el patrón que debe cumplir cada el nombre de cada uno de sus módulos.

El problema que plantea este método es que no se puede garantizar que la intersección del conjunto de valores posibles para cada una de las máscaras sea un conjunto vacío.

Por ejemplo: Los módulos de una aplicación de Sistemas pueden tener como criterio de nomenclatura el patrón: S\* y a su vez una aplicación de clasificación de clientes podría utilizar el patrón SRT\*, con lo que, en principio, el módulo SRTB0101 podría pertenecer a la aplicación de sistemas o a la de clasificación.

Para evitar esta situación se realiza el siguiente proceso:

* Por defecto se crea la aplicación General con el patrón “\*” que cubre cualquier nombre y garantiza que todo módulo pertenece a una aplicación.
* En el momento de dar de alta un nuevo registro se aplica un algoritmo que establece una valores numéricos a los campos *fijo* y *peso* en función del número de caracteres fijos y especiales presentes en la máscara, dando más peso a aquellas que son más específicas.

En el momento de establecer la asociación se procesan las máscaras por orden descendente de su peso y ascendente del valor del campo fijo y se aplica la primera mascara que aplique.

Existe una situación especial, que se produce con el conjunto de máscaras que no tienen caracteres especiales, por ejemplo, SRTB0101 o SRTAMQS. Este conjunto de valores pueden generar los mismos valores para los campos *fijo* y *peso* si contienen el mismo número de caracteres y se procesan directamente mediante una consulta directa a la tabla.

Siguiendo con el ejemplo anterior, para asociar la aplicación correcta a los módulos SRTB0101 y SAUD3245 primero se pregunta si existe una máscara que sea exactamente igual al nombre del módulo.

Si no existe se recuperaran las máscaras que no definen nombres concretos que, de acuerdo con el criterio de ordenación indicado devolverá primero SRT\* y luego S\* puesto que la segunda máscara es menos especifica que la primera y asociará SRTB0101 a la aplicación de clasificación y SAUD3245 a la de Sistemas puesto que SAUD3245 no satisface la máscara SRT\*.

## Tablas de módulo

Son las tablas que contienen la información estática de una versión de un módulo.

Una versión de un módulo se establece de manera automática en base a la firma digital del código fuente; es decir, en el momento en que en el código fuente se modifique un byte, su firma digital variará y se generará una nueva versión de ese módulo.

Las tablas que forman este grupo son:

|  |  |
| --- | --- |
| Tabla | Descripción |
|  |  |
| MOD\_BADSTMTS | Contiene la información de las instrucciones no permitidas |
|  |  |
| MOD\_BLOQUES | Contiene la lista de todos los bloques de ejecución del módulo |
|  |  |
| MOD\_DEPENDENCIAS | Contiene las referencias a las COPYS y módulos utilizados por el programa |
|  |  |
| MOD\_FICHEROS | Contiene la información de los ficheros utilizados en el módulo |
|  |  |
| MOD\_FUENTE | Contiene el código fuente comprimido en formato ZIP |
|  |  |
| MOD\_GRAFO | Contiene la información necesaria para construir el grafo del módulo |
|  |  |
| MOD\_PARRAFOS | Contiene la lista de párrafos |
|  |  |
| MOD\_REL\_PARRAFOS | Contiene la relación (llamadas) entre los párrafos |
|  |  |
| MOD\_RESUMEN | Contiene la información relativa a los contadores de número de líneas de código, sentencias, comentarios, etc. |
|  |  |
| MOD\_SECCIONES | Contiene los indicadores relativos a cada una de las secciones del programa |
|  |  |
| MOD\_VERSIONES | Contiene la relativa a la versión del módulo; su firma, nombre, … |

Tabla 17: Tablas MOD

El diagrama de relación entre este conjunto de tablas se muestra a continuación:



Diagrama 12: E/R Grupo tablas MOD

## Tablas de ejecución

Son las tablas que contienen la información generada durante la ejecución de un programa por el componente Trapper. Cada tabla guarda un tipo de mensaje

Las tablas que forman este grupo son:

|  |  |
| --- | --- |
| Tabla | Descripción |
|  |  |
| TRP\_CALL | Contiene los mensajes de tipo CALL |
|  |  |
| TRP\_COBERTURA | Contiene los mensajes de cobertura de código |
|  |  |
| TRP\_ MODULOS | Contiene los mensajes de módulos ejecutados |
|  |  |
| TRP\_PARRAFOS | Contiene los mensajes de párrafos ejecutados |
|  |  |
| TRP\_ PARR\_WORKING | Contiene la información de uso de los párrafos |
|  |  |
| TRP\_ PERFORM | Contiene los mensajes de tipo PERFORM |
|  |  |
| TRP\_ PERSISTENCIA | Contiene la información acerca del uso de los ficheros |
|  |  |
| TRP\_SESIONES | Contiene la información de la sesión |

Tabla 18: Tablas TRP

Todas las tablas dependen de la versión del módulo asociada a esa ejecución. El diagrama de relación entre este conjunto de tablas se muestra a continuación:

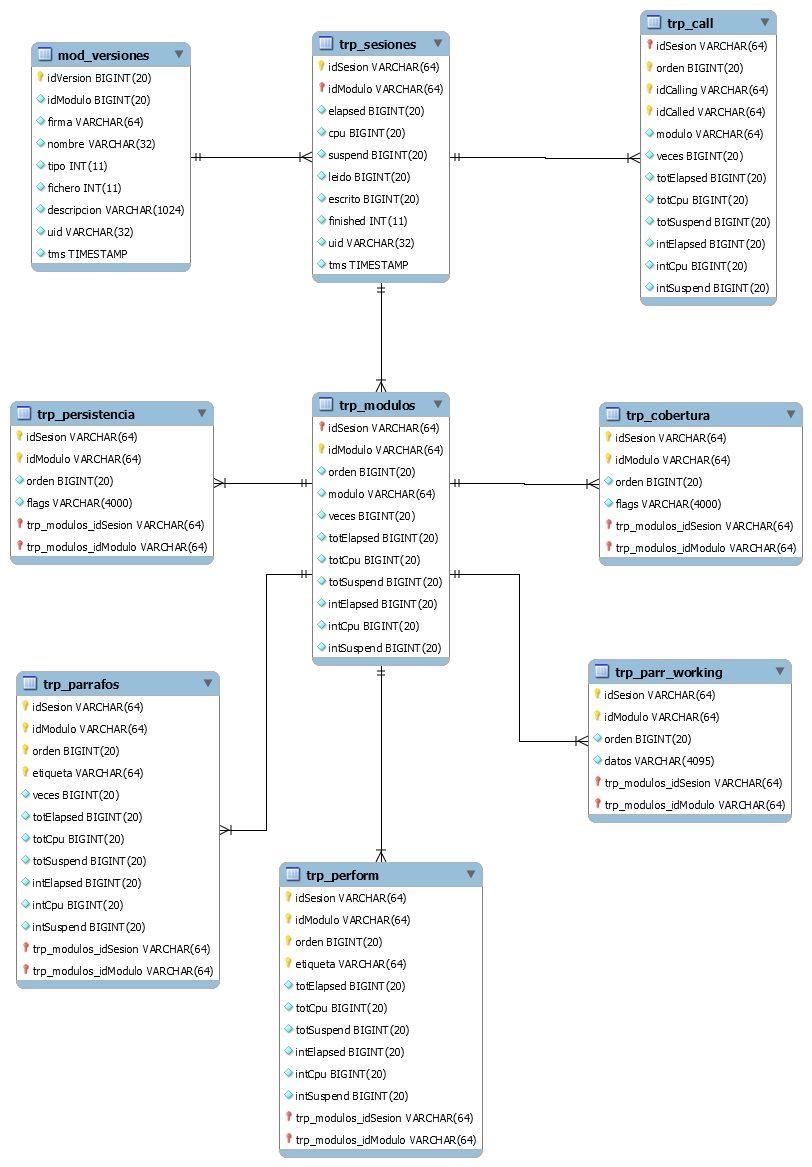


Diagrama 13: E/R Grupo de tablas TRP

## Tablas de sesión

Son las tablas que contienen la información consolidada a partir de la información generada durante la ejecución de un programa por el componente Trapper.

Las tablas que forman este grupo son:

|  |  |
| --- | --- |
| Tabla | Descripción |
|  |  |
| SES\_ARBOL | Contiene la información relativa al árbol de llamadas |
|  |  |
| SES\_IO | Contiene la información acumulada de los accesos a los ficheros |
|  |  |
| SES\_MODULOS | Contiene la información acumulada de los consumos de cada módulo |
|  |  |
| SES\_PARRAFOS | Contiene la información acumulada de los consumos de los párrafos |
|  |  |
| SES\_PERSISTENCIA | Contiene la información del uso de los ficheros del módulo |
|  |  |
| SES\_SESIONES | Contiene la información de la sesión |

Tabla 19: Tablas SES

Todas las tablas dependen de la versión del módulo asociada a esa ejecución y de la sesión de la tabla TRP\_SESION. El diagrama de relación entre este conjunto de tablas se muestra a continuación:

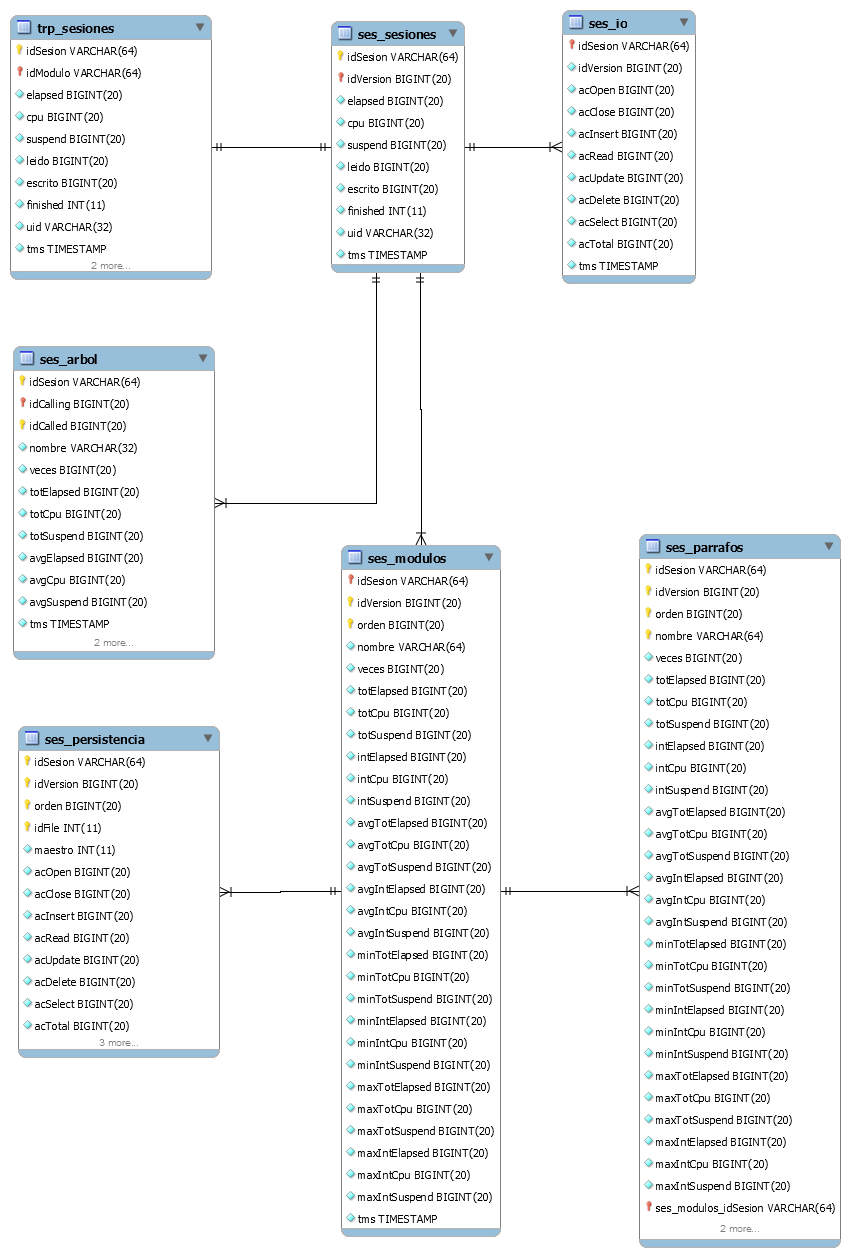


Diagrama 14: E/R Grupo tablas SES

## Tablas sumarizadas

Son las tablas que contienen la información consolidada de todas las sesiones ejecutadas. Se alimentan a través de la información contenidas en las tablas de sesión

Las tablas que forman este grupo son:

|  |  |
| --- | --- |
| Tabla | Descripción |
| Tabla 20: Tablas SUM |  |
| SUM\_ARBOL | Contiene la información relativa a las llamadas entre módulos |
|  |  |
| SUM\_MODULOS | Contiene la información de consumos de cada módulo |
|  |  |
| SUM\_PARRAFOS | Contiene la información de consumos de cada párrafo |

El diagrama de relación entre este conjunto de tablas se muestra a continuación:

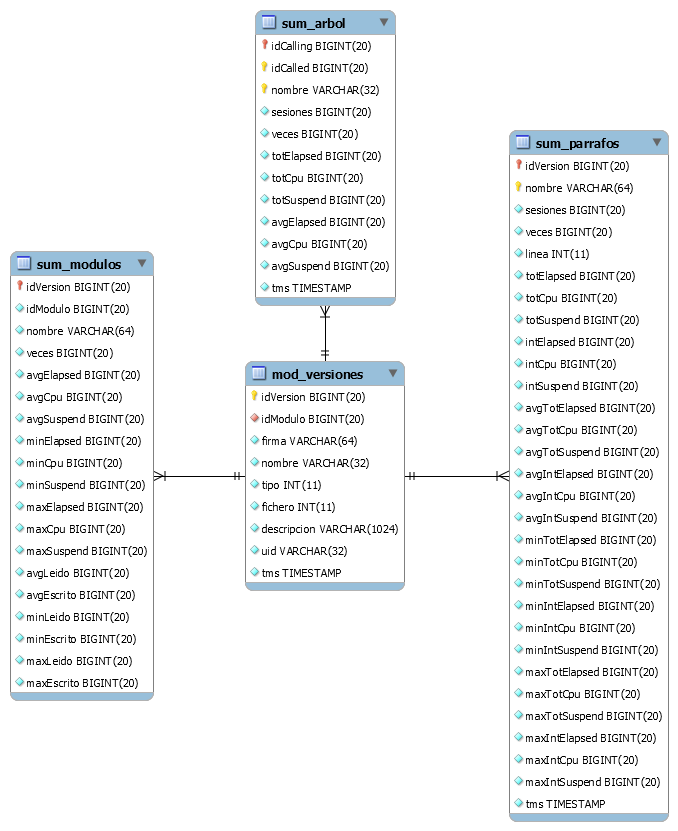


Diagrama 15: E/R Grupo de tablas SUM

## Tablas de *Logging*

Son las tablas que contienen la información de los diferentes eventos que se han producido en el sistema

Las tablas que forman este grupo son:

|  |  |
| --- | --- |
| Tabla | Descripción |
|  |  |
| LOG\_LOGGING | Contiene la información necesaria para generar un mensaje |
|  |  |
| LOG\_MSG | Contiene los diferentes mensajes que se pueden generar en cada uno de los idiomas soportados |
|  |  |
| LOG\_TIPO | Contiene los textos asociados al tipo de mensaje generado de acuerdo con los idiomas soportados: Aviso, Info, Error, etc. |

Tabla 21: Tablas LOG

Debido a la localización de los mensajes, la tabla de mensajes LOG\_LOGGING no puede almacenar mensajes si no los códigos y la información necesaria para la generación del mensaje bajo petición de acuerdo con el idioma en uso.

El diagrama de relación entre este conjunto de tablas se muestra a continuación:

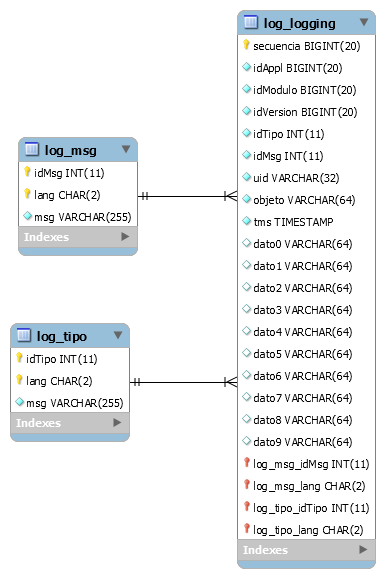


Diagrama 16: E/R tablas grupo LOG

# Instalación y configuración

Es esta capítulo se describen los procedimientos necesarios para la instalación y configuración del sistema en base a las siguientes arquitecturas hardware y software

## Arquitectura hardware

Para el servidor del sistema se ha utilizado una distribución de Linux Ubuntu con la siguiente configuración

|  |  |
| --- | --- |
| Diagrama 17: Arquitectura Hardware | **/ - Disco ubuntu.vmdk**  Contiene el sistema con el software necesario para su ejecución.  **/PFC – Disco PFC.vmdk**  Contiene el software, el código y la documentación del proyecto.  En caso de querer ejecutar el sistema en otra plataforma Linux, este disco es el único requerido si la plataforma cumple con los requerimientos de software establecido |
| **/software – Disco software.vmdk**  Contiene el software requerido que no está disponible en los repositorios de software de Ubuntu.  Una vez instalado el software este disco no es requerido y se puede desmontar  **Swap – Disco swap.vmdk**  Dispositivo de paginación. Por motivos de rendimiento es recomendable en un disco separado del sistema operativo. Por otro lado, cuando se utilizan diferentes máquinas virtuales de manera no simultánea, este disco puede ser compartido por todas ellas. | |

## Arquitectura Software

Desde el punto de vista de la arquitectura software, la configuración utilizada se muestra en el siguiente diagrama:



Diagrama 18: Arquitectura Software

|  |  |
| --- | --- |
| Componente | Descripción |
|  |  |
| Sistema Operativo | Distribución Linux Ubuntu Server 15.0.4 para una arquitectura de procesador AMD 64 bits |
|  |  |
| Servidor de base de datos | MySQL Server versión 14.14  La base de datos utilizada se define como **SDP** |
|  |  |
| Sistema de mensajería | Websphere MQ Series 8.0.2 |
|  |  |
| Servidor de páginas | Apache 2 |
|  |  |
| Servidor de aplicaciones | Tomcat Apache 8 |

Tabla 22: Componentes software

## Servidor

Para mantener el sistema únicamente con el software necesario se han instalado los siguientes paquetes desde el repositorio central de Ubuntu evitando instalar otros paquetes asociados con el comando:

apt-get install –no-install-recommends *nombre\_del\_paquete*

| Paquete | Contenido |
| --- | --- |
|  |  |
| openSSH | Shell Seguro. Se instala durante la instalación del sistema |
|  |  |
| Vsftpd | Servidor ftp |
|  |  |
| Gcc | Compilador GNU C |
|  |  |
| rpm | Gestor de paquetes RPM |
|  |  |
| Make | Utilidad para gestión de compilaciones |
|  |  |
| automake | Herramienta para la generación de Makefiles conformes con los estándares GNU |
|  |  |
| autoconf | Herramienta de generación automática de scripts de configuración |
|  |  |
| default-jre | Open Java Runtime Environment |
|  |  |
| default-jdk | Open Java Development Toolkit |
|  |  |
| open-cobol | Compilador Open Cobol |
|  |  |
| apache2 | Servidor de páginas Apache 2 |
|  |  |
| tomcat8 | Servidor de aplicaciones Apache Tomcat 8 |
|  |  |
| tomcat8-admin | Aplicación de administración Web de Apache Tomcat 8 |
|  |  |
| tomcat8-user | Herramienta para crear instancias de usuarios en Tomcat 8 |
|  |  |
| libapache2-mod-jk | Conector entre Apache y Tomcat |
|  |  |
| mysql-server | Servidor de base de datos MySQL |
|  |  |

Tabla 23: Paquetes a instalar

Los paquetes que no están disponibles en el repositorio central o que necesitan un tratamiento diferenciado están incluidos en el disco **/software**. Una vez realizada la instalación, este disco podría ser desmontado del sistema.

### Sistema de mensajería WebSphere MQ Series

Los paquetes relativos a MQSeries están empaquetados en formato Debian por lo que pueden mensajes de error en Ubuntu, para evitarlo estos paquetes se deben ejecutar con la orden:

rpm -ivh --nodeps --force-debian paquete\_mq.rpm

| Paquete | Contenido |
| --- | --- |
|  |  |
| mqm/mqlicense.sh | Ejecutar con el parámetro **-accept** para iniciar la instalación |
|  |  |
| mqm/MQSeriesRuntime | Funciones comunes a MQSeries |
|  |  |
| mqm/MQSeriesServer | Servidor MQSeries |
|  |  |
| mqm/MQSeriesClient | Cliente de MQSeries C |
|  |  |
| mqm/MQSeriesJava | API Java y JMS |
|  |  |
| mqm/MQSeriesJRE | Java Runtime |
|  |  |
| mqm/MQSeriesSDK | Librerías y ficheros de cabecera para C |
|  |  |
| mqm/MQSeriesMsg\_es | Mensajes en castellano |
|  |  |

Tabla 24: Paquetes MQ Series a instalar

## Configuración

### Servidor

#### Usuarios

Durante la instalación del servidor y de los diferentes paquetes se han creado los siguientes usuarios y asignadas las siguientes contraseñas:

| Usuario | Tipo | Contraseña |
| --- | --- | --- |
|  |  |  |
| root | Administrador | Uned |
|  |  |  |
| jgonzalez373 | Usuario | uned |
|  |  |  |
| tomcat8 | Administrador tomcat | tomcat8 |
|  |  |  |
| Mqm | Administrador MQ | Mqm |
|  |  |  |
|  |  |  |
|  |  |  |

Tabla 25: Usuarios servidor

Y los siguientes grupos

| Grupo | Descripción |
| --- | --- |
|  |  |
| mqm |  |
|  |  |
| tomcat8 |  |
|  |  |
| pfc | Grupo al que pertenecen los usuarios del sistema |
|  |  |
|  |  |

### MQSeries

En este apartado se indican los pasos a realizar para configurar el sistema de gestión de colas de acuerdo con los requerimientos del sistema y definir los siguientes objetos:

|  |  |
| --- | --- |
| Objeto | Definición |
| SDP.QMGR | Gestor de colas |
| SDP.INIT | Cola de inicialización |
| SDP.PARSER | Cola para los mensajes generados por SDPParser |
| SDP.TRAPPER | Cola para los mensajes generados por SDPTRAPB |
| SDP.COLLECTOR | Cola de sincronización interna |
| PRC.PARSER | Proceso que se ejecutará cuando existan mensajes en la cola SDP.PARSER |
| PRC.TRAPPER | Proceso que se ejecutará cuando existan mensajes en la cola SDP.TRAPPER |
| PRC.COLLECTOR | Proceso que se ejecutará cuando existan mensajes en la cola SDP.COLLECTOR |

Tabla 26: Objetos de mensajeria

El proceso requiere los siguientes pasos:

1. Configurar el usuario autorizado
2. Establecer la instalación principal
3. Crear el gestor
4. Crear los objetos
5. Establecer los permisos
6. Configurar el sistema en tiempo de arranque

#### MQSeries

Por defecto este paquete se instala en /opt/mqm y crea el usuario mqm y el grupo mqm.

Se le establece la contraseña de ese usuario a mqm

Este el usuario que tiene los permisos necesarios para administrar el MQ

#### Establecer la instalación principal

MQSeries permite que coexistan varias instalaciones en el mismo servidor pero debe haber una de hecha definida como principal.

Para verificar si se ha creado por defecto ejecutar el comando:

/opt/mqm/bin/dspmqinst

Si no existe una instalación principal, crearla con el comando:

/opt/mqm/bin/setmqinst /opt/mqm

#### Creación del gestor de colas

Se debe crear un gestor de colas para el sistema, preferentemente como gestor de colas predeterminado ya que en caso contrario será necesario indicarlo de manera explícita en todas las conexiones.

/opt/mqm/bin/crtmqm -q SDP.QMGR

Una vez creado, se arranca manualmente para continuar con el proceso:

/opt/mqm/bin/strmqm SDP.QMGR

#### Creación de los objetos

Una vez iniciado el gestor de colas, se deben crear los objetos indicados en la tabla Tabla 26: Objetos de mensajeria. Esto se puede realizar manualmente o utilizando el script de configuración aportado en el anexo: XXXXXX mediante el siguiente comando, ejecutado como usuario mqm:

/opt/mqm/bin/runmqsc < mqseries.cfg

#### Establecimiento de los permisos

Es necesario otorgar los permisos necesarios a los diferentes usuarios para conectarse al sistema y acceder a los objetos.

Por simplicidad, los permisos se establecerán para el grupo **pfc** de manera que cualquier usuario de ese grupo pueda acceder al sistema de mensajería.

Los comandos a ejecutar son:

setmqauth –m SDP.QMGR –t qmgr –n SDP.QMGR –g pfc +connect

setmqauth –m SDP.QMGR –t queue –n SDP.\* -g pfc +all

setmqauth –m SDP.QMGR –t queue –n SDP.\* -g pfc +allmqi

Setmqauth –m SDP.QMGR –t process –n PRC.\* -g pfc +all

Que establece las autorizaciones para, respectivamente conectarse al servidor, usar las colas SDP en modo servidor y cliente, ejecutar los procesos PFC

#### Automatización del arranque

El gestor de colas se arranca manualmente mediante la orden strmqm, pero dado que se considera que es una parte del sistema, resulta mas conveniente prepararlo para que se inicie cada vez que se inicie el sistema.

Para ello es necesario incluir un script de inicio en la lista de procesos que se deben arrancar al inicio del sistema en el directorio etc/init.d

Este script de configuración se detalla en el anexo XXXX

Una vez instalado el script se notifica al proceso de arranque con el comando, ejecutado como usuario root:

update-rc.d mq defaults 95 02

Donde:

mq es el nombre dado al script

95 y 02 son ordenes relativos para los procesos de arranque y parada del script.

### Servidores de páginas y de aplicaciones

En este apartado se configura el servidor de páginas y de aplicaciones basados en apache2 y tomcat permitiendo que se comuniquen entre ellos mediante una configuración básica.

* Para establecer otras configuraciones u obtener información acerca de esta consultar “The AJP Connector” [AJP]
* “Apache Tomcat Configuration Reference” [TOMC]
* “Apache HTTP Server Project” [APACHE]

Los pasos a realizar son:

1. Configurar el servidor de aplicaciones
2. Configurar un servidor de páginas virtual

Los archivos de configuración indicados en este apartado se encuentran disponibles en el disco XXXXX

#### Configurar el servidor de aplicaciones

Activar el conector AJP 1.3 descomentando en el fichero etc/tomcat8/server.xml

la línea:

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

Crear el fichero de configuración workers.properties. Un ejemplo de este fichero está disponible en XXXX

worker.list=sdp

worker.sdp.port=8009

worker.sdp.host=localhost

worker.sdp.type=ajp13

Código 7: Ejemplo de fichero workers.properties

y guardarlo en /etc/libapache2-mod-jk

Para poder acceder a la consola de administración es necesario definir los usuarios que pueden acceder a ella, para ello es necesario modificar el fichero /etc/tomcat8/tomcat-users.xml de la siguiente manera:

<tomcat-users>

<role rolename="tomcat"/>

<role rolename="admin"/>

<role rolename="manager"/>

<role rolename="role1"/>

<role rolename="manager-gui"/>

<role rolename="manager-script"/>

<role rolename="manager-jmx"/>

<role rolename="manager-status"/>

<user username="tomcat" password="tomcat" roles="tomcat,manager-gui,manager-status" />

<user username="jgonzalez373" password="jgg"

roles="tomcat,manager-gui,manager-status, manager-script, manager-jmx" />

</tomcat-users>

Código 8: Ejemplo tomcat-users.xml

#### Configurar el servidor de páginas

Definir el host virtual en /etc/apache2/sites-available/sdp.conf:

<VirtualHost \*:80>

ServerAdmin javier.gonzalez.grandez@gmail.com

ServerName www.sdp.com

DocumentRoot "/PFC/web/sdp"

<Directory "PFC/web/sdp">

Options Indexes FollowSymLinks Includes ExecCGI

AllowOverride All

Require all granted

</Directory>

<IfModule dir\_module>

DirectoryIndex index.shtml index.html index.htm

</IfModule>

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

LogLevel info

# Conector a tomcat

JkMount /sdp\* sdp\_worker

</VirtualHost>

Código 9: Virtual Host SDP.conf

Habilitarlo en el servidor apache:

a2ensite sdp.conf

### Servidor de base de datos

El servidor de base de datos ya se ha configurado a sus valores por defecto durante el proceso de instalación.

El único paso pendiente es autorizar a los usuarios a conectarse a la base de datos:

CREATE USER 'jgonzalez373'@'%' IDENTIFIED BY 'uned';

GRANT ALL PRIVILEGES ON \*.\* TO 'jgonzalez373'@'%' WITH GRANT OPTION;

Código 10: Autorizar usuario en MySQL

# Manual de usuario

## Parámetros y configuración

Con excepción del componente Web, el resto de componentes del sistema se pueden ejecutar desde una consola y aceptar un conjunto de parámetros que definirán el comportamiento en esa instancia.

Estos parámetros se pueden especificar de tres maneras:

1. A través de un fichero de configuración
2. Mediante una variable de entorno
3. En la propia línea de comandos

La secuencia en la que se aplica la configuración se realiza de acuerdo con el siguiente orden:

1. Los valores por defecto establecidos en el programa
2. Las posibles variables de entorno definidas
3. Un fichero de configuración si se ha indicado en la línea de comandos
4. Las opciones establecidas de manera específica en la línea de comandos

### Fichero de configuración

El fichero de configuración es un fichero de texto que asigna a un parámetro un valor, con las siguientes características:

* Se establece una opción por línea en la forma clave = valor
* Las espacios en blanco no son significativos
* Se puede indicar un comentario con el carácter especial ‘#’ que aplica hasta el final de la línea

### Variables de entorno

Se definen a continuación las variables de entorno soportadas por el sistema

| **Variable** | **Acción** |
| --- | --- |
|  |  |
| SDP\_INPUT |  |
|  |  |
| SDP\_OUTPUT |  |
|  |  |
| SDP\_MARGIN\_LEFT |  |
|  |  |
| SDP\_MARGIN\_RIGHT |  |
|  |  |
| SDP\_QUEUE |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## SDPParser

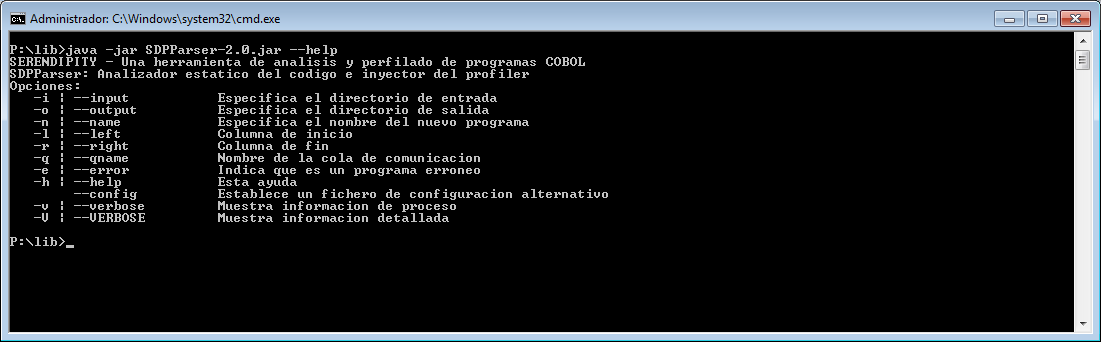
Procesa los programas COBOL pasados por línea de comandos y:

* Genera, por cada uno de ellos, otro programa funcionalmente equivalente con información de perfilado incluida
* Envía la información analítica del programa origen al servidor

Uso: java –jar SDPParser [opciones] lista\_programas\_fuente

Opciones:

|  |  |  |
| --- | --- | --- |
| -e  --error |  | Indica que se debe procesar un programa incorrecto.  Se debe utilizar cuando se ha producido un error al compilar el programa generado |
|  |  |  |
| -h  --help |  | Muestra la ayuda relativa al componente |
|  |  |  |
| -v  --verbose |  | Muestra información de progreso |
|  |  |  |
| -V |  | Muestra información más detallada del progreso |
|  |  |  |
| --config | config.properties | Especifica un fichero de configuración alternativo |
|  |  |  |
| -i  --input | directorio | Especifica el directorio de entrada por defecto donde se buscará el fichero a procesar si este no es una ruta absoluta |
|  |  |  |
| -o  --output | directorio | Especifica el directorio donde se guardará el nuevo programa generado. |
|  |  |  |
| -n  --name | nombre\_archivo | Especifica un nombre para el nuevo programa generado.  Este parámetro aplica únicamente en los casos en los que se procesa un único programa fuente |
|  |  |  |
| -l  --left | numero | Especifica donde comienza la zona A (margen izquierdo) del programa fuente |
|  |  |  |
| -r  --right | numero | Especifica la columna límite del fichero fuente |
|  |  |  |
| -q  --qname | nombre\_cola | Especifica el nombre de la cola que se usará para el envío de la información de análisis |



openSuSe 13.2 32 bits

disco swap /dev/sda1

discoduro /dev/sda2

Usuarios:

Jgonzalez373:jgg

<http://www.muycomputerpro.com/2015/03/26/demanda-expertos-cobol-universidad-formacion>

<http://www.infoweek.biz/la/2015/04/crece-demanda-expertos-cobol/>

<https://scs.senecac.on.ca/~timothy.mckenna/offline/COBOL_not_dead_yet.htm>

Guia de instalacion MQSeries

<http://www-01.ibm.com/support/docview.wss?uid=swg21656068>

1. En determinadas arquitecturas es posible que un programa se vaya ejecutando en diferentes procesadores e incluso en diferentes máquinas a lo largo de su ciclo de vida, pero se mantiene la condición de que, en un instante dado, el proceso solo utiliza un procesador y su información de auditoria (consumo de CPU y de recursos) es consistente a lo largo de esas máquinas y procesadores. [↑](#footnote-ref-2)
2. Esta situación es típica en procesos de acceso a Bases de Datos [↑](#footnote-ref-3)