ARAF-OOP Method — Modelo Ardiles

Innovando con RPG: De la Tradición a la Modernidad

Un enfoque práctico en AS/400 con DB2/400

Versión 1.0 — Octubre 2025

Jesús Ardiles – Senior Specialist in AS/400 Systems
& IBM Certified in Architecture Thinking
✨ “Innovando con RPG desde la tradición hacia la modernidad”

Contenido

Capítulo 1 – Introducción y Contexto

El lenguaje RPG ha acompañado durante décadas a los desarrolladores en la plataforma IBM i (AS/400), consolidándose como un estándar de robustez, estabilidad y cercanía con los procesos de negocio. Sin embargo, la evolución de los paradigmas de programación y la necesidad de integrar sistemas con arquitecturas modernas han impulsado la transformación del RPG hacia un enfoque más flexible y estructurado.

En este camino surge la Programación Orientada a Objetos (OOP) como una técnica que, lejos de reemplazar lo aprendido, permite aprovechar la solidez de RPG mientras se incorporan prácticas modernas de diseño.

Lo sorprendente es que la transición no es abrupta: bastaron dos días de estudio y práctica para descubrir que los conceptos de objetos, clases y métodos pueden adaptarse naturalmente al ecosistema IBM i. Al tercer día, esta metodología ya se aplicaba en un proyecto real, confirmando que la modernidad puede integrarse a la tradición de manera orgánica.

A partir de esta experiencia práctica se desarrolla el contenido del tutorial que sigue.

Capítulo 2 – Conceptos base de OOP en RPG

La Programación Orientada a Objetos (OOP) se fundamenta en la idea de representar entidades del mundo real o del negocio como clases que definen atributos y métodos que encapsulan su comportamiento. A partir de esas clases se instancian objetos, que interactúan entre sí para dar vida a la lógica de la aplicación.

Los pilares de este paradigma son:
- Encapsulación: agrupar atributos y métodos en una misma entidad.

- **Abstracción**: modelar solo lo esencial de un objeto, ocultando detalles innecesarios.

- Herencia: permitir que una clase herede características de otra, facilitando la reutilización.

- **Polimorfismo**: un mismo método puede comportarse de distintas maneras según el objeto que lo invoque.

 # Adaptación al RPG Free en IBM i
RPG, en su evolución hacia la versión ILE RPG Free-Format, incorporó elementos que permiten trabajar con estructuras más cercanas a la OOP, aunque con un enfoque particular al ecosistema IBM i:
- **Ctl-Opt**: define las opciones globales del programa, actuando como el 'contexto' en que los objetos existen.
- Definición de pantallas (**Workstn**): las pantallas pueden considerarse objetos gráficos, con indicadores que se manipulan como atributos de clase.
- Estructuras de Datos (**Dcl-Ds**): permiten modelar registros de bases de datos o pantallas como entidades equivalentes a objetos.
- Procedimientos (**Dcl-Proc**): funcionan como métodos, encapsulando la lógica de interacción.
De este modo, en RPG Free podemos pensar cada pantalla como una clase, cada registro de datos como un objeto y cada procedimiento como un método.

 # Ejemplo inicial
Veamos un fragmento en RPG Free donde definimos el contexto global y la pantalla principal como objeto:

```
Ctl-Opt  OPTION(*SRCSTMT:*NODEBUGIO:*NOUNREF:*SECLVL)
         DatFmt(*iso) TimFmt(*iso) AlwNull(*usrctl) DecEdit('0.')
         Fixnbr(*Zoned) Aut(*All) datedit(*ymd) BNDDIR('QC2LE')
         Main(AUD024R_Main);

// Definición de pantalla como objeto
Dcl-F Display workstn(*ext) qualified
                 extdesc('AUD0240D')
                 extfile(*extdesc)
                 sfile(SFL01: RRN01)
                 indds(WsInd)
                 usropn;
```

Capítulo 3. Arquitectura Interna y Procedimientos

3.1 Arquitectura interna del programa

La arquitectura del programa RPG OOP se organiza en torno a procedimientos que interactúan con las pantallas, gestionan la lógica de negocio y acceden a la base de datos DB2/400.

3.2 Procedimiento Driver

El procedimiento Driver actúa como el núcleo del flujo, gestionando la interacción con la pantalla principal, el control de indicadores y la invocación de procedimientos auxiliares.

3.3 Procedimiento LoadOnePage

LoadOnePage encapsula la lógica para cargar datos paginados desde DB2/400 hacia la subfile de pantalla. Este procedimiento demuestra la naturaleza orientada a objetos del enfoque.

3.4 Equivalencia con un lenguaje moderno (Java)

Podemos comparar el procedimiento LoadOnePage en RPG con un método en Java que realiza la misma función: cargar datos en un objeto lista e iterarlos en pantalla. Esta comparación evidencia la cercanía de RPG moderno con paradigmas actuales.

3.5 Conexión con la práctica

A partir de esta experiencia práctica se desarrolla el contenido del tutorial que sigue, mostrando cómo RPG puede aprovechar conceptos de orientación a objetos en la vida real.

Capítulo 4. El Procedimiento Driver en detalle

A continuación, se presenta el código completo del procedimiento Driver, acompañado de comentarios explicativos sobre su rol en la arquitectura interna:

```
Dcl-Proc  Driver;

  Dcl-Ds  SFL01_rec    Likeds(SFL01_rec_t)    inz;

  Dcl-Ds  DataArray  Qualified    Dim(SFL01_PAGESIZE);
    useraud     char  (10);
    fname       char  (20);
    typuser     char  (10);
    lname       char  (20);
    status      char  (01);
  End-Ds;

  Dcl-Ds  CTL01_rec         likeds(CTL01_rec_t)     inz;
  Dcl-Ds  SCREEN01_rec      likeds(SCREEN01_rec_t)  inz;
  Dcl-Ds  SCREEN02_rec      likeds(SCREEN02_rec_t)  inz;
  Dcl-Ds  SCREEN03_rec      likeds(SCREEN03_rec_t)  inz;

  Dcl-S  Size01             like(RRN01);
  Dcl-S  String             varchar(512);
  Dcl-S  EndOfData          ind;
  Dcl-S  SaveNivInsp        like(CTL01_rec.nivinsp);
  Dcl-S  SaveSortOption     like(CTL01_rec.SortOption);
  Dcl-S  ioUserAud          char(10)  inz          ;
  Dcl-S  ioUserStat         char(1)   inz         ;

  Dcl-Pr AUD025R;
    *n Like(DsTrama) Const;
  End-Pr AUD025R;

   CTL01_rec.SortOption = SortByUser;
   CTL01_rec.Nivinsp = *blanks;
   CTL01_REC.TITLE = c_Tittle;
   Dstrama.userEven = PgmDs.Nombre_Usr;
   DsTrama.PgmEven = PgmDs.Nombre_Progr;
```

```
   DsTrama.AppEven = 'AUDILOTES';

   Dow '1';
      If CTL01_rec.SortOption <> SaveSortOption Or
         CTL01_rec.nivinsp <> SaveNivinsp;
         LoadFirstPage (CTL01_rec: Size01: EndOfData: SCREEN01_rec);
         CTL01_rec.SFLRCDNBR = 1     ;
         SaveSortOption = CTL01_rec.SortOption;
         SaveNivinsp = CTL01_rec.nivinsp;
      Endif;

      WsInd.SflDspCtl  =  *on;
      WsInd.SflDsp     =  (Size01 > *zero);
      WsInd.sflClr     =  *off;
      WsInd.sflEnd     =  EndOfData;

      WRITE   Display.Screen01  Screen01_rec;
      EXFMT   Display.CTL01     CTL01_rec;
      If WsInd.ExitKey;                       // Si presiona F3 Salir
         leave;
      Endif;

      Clear Screen01_rec.Message;
      If WsInd.RollUp;                       // Avance de Pagina
         LoadOnePage (Size01: EndOfData: SCREEN01_rec);
         CTL01_rec.SFLRCDNBR = Size01;
      Endif;

      If WsInd.AddReg ;                      // si presiona F6 Agregar
Registro
         LoadValuesPage (Size01: EndOfData: CTL01_rec: SFL01_rec);
         Clear CTL01_rec.SortOption;
      Endif;

      if CTL01_rec.C1RRN > *zeros;          // Si dio doble CLICK sobre estado
                                            // o nombre de usuario para
                                            // cambiar estado
         chain CTL01_rec.C1RRN Display.SFL01 SFL01_rec  ;
         if %found;
            SCREEN03_rec.USERAUD = SFL01_rec.USERAUD;
            DsTrama.DescEven = '*CambioEstatus-'+
                               SFL01_rec;
            ioUserAud = SFL01_rec.USERAUD;
            EXFMT   Display.SCREEN03  SCREEN03_rec;
            If WsInd.Confirma;
               Select;
                  When SFL01_rec.STATUS = 'A';
```

```
                    ioUserStat ='D';
                    SFL01_rec.STATUS = ioUserStat;

                When SFL01_rec.STATUS = 'D';
                    ioUserStat ='A';
                    SFL01_rec.STATUS = ioUserStat;
             EndSl;
             Exec sql UPDATE AUDUSERPF A SET STATUS = :ioUserStat
              WHERE A.USERAUD = :ioUserAud;
              if SQLSTT = '00000';
                 DsTrama.DescEven = %TRIM(DsTrama.DescEven)+'> '+
                                    '*CmbioEstatus-'+SFL01_rec;
                 AUD025R(DsTrama);
              Endif;
             Clear CTL01_rec.SortOption;
          Endif;
       Endif;
    Endif;
  Enddo;

End-Proc  Driver;
```

Análisis y explicación

1. **Bucle principal (dow not EndProgram)**
    - Garantiza que la aplicación permanezca activa hasta que el usuario decida salir (tecla de salida).
    - Actúa como el *game loop* en un motor de videojuegos: evalúa estado, recibe entradas y redibuja pantallas.
2. **Gestión de eventos con select/when**
    - Cada **tecla de función** o indicador de pantalla es interpretada como un *evento*.
    - Se asignan rutas claras para cada opción: listar registros, agregar, modificar, salir.

3. **Integración con otros procedimientos**
    - El Driver() **no carga datos directamente**, sino que delega:
        - LoadOnePage() para cargar y mostrar registros.
4. Uso de exfmt para interactuar con pantallas específicas (SCREEN01, SCREEN02, SCREEN03).
5. **Separación de responsabilidades**
    - El núcleo de control (Driver) no se mezcla con lógica de negocio ni SQL, manteniendo la arquitectura limpia.
    - Esto facilita que nuevas pantallas o eventos puedan añadirse sin alterar la estructura base.

4.1 UML del procedimiento DRIVE

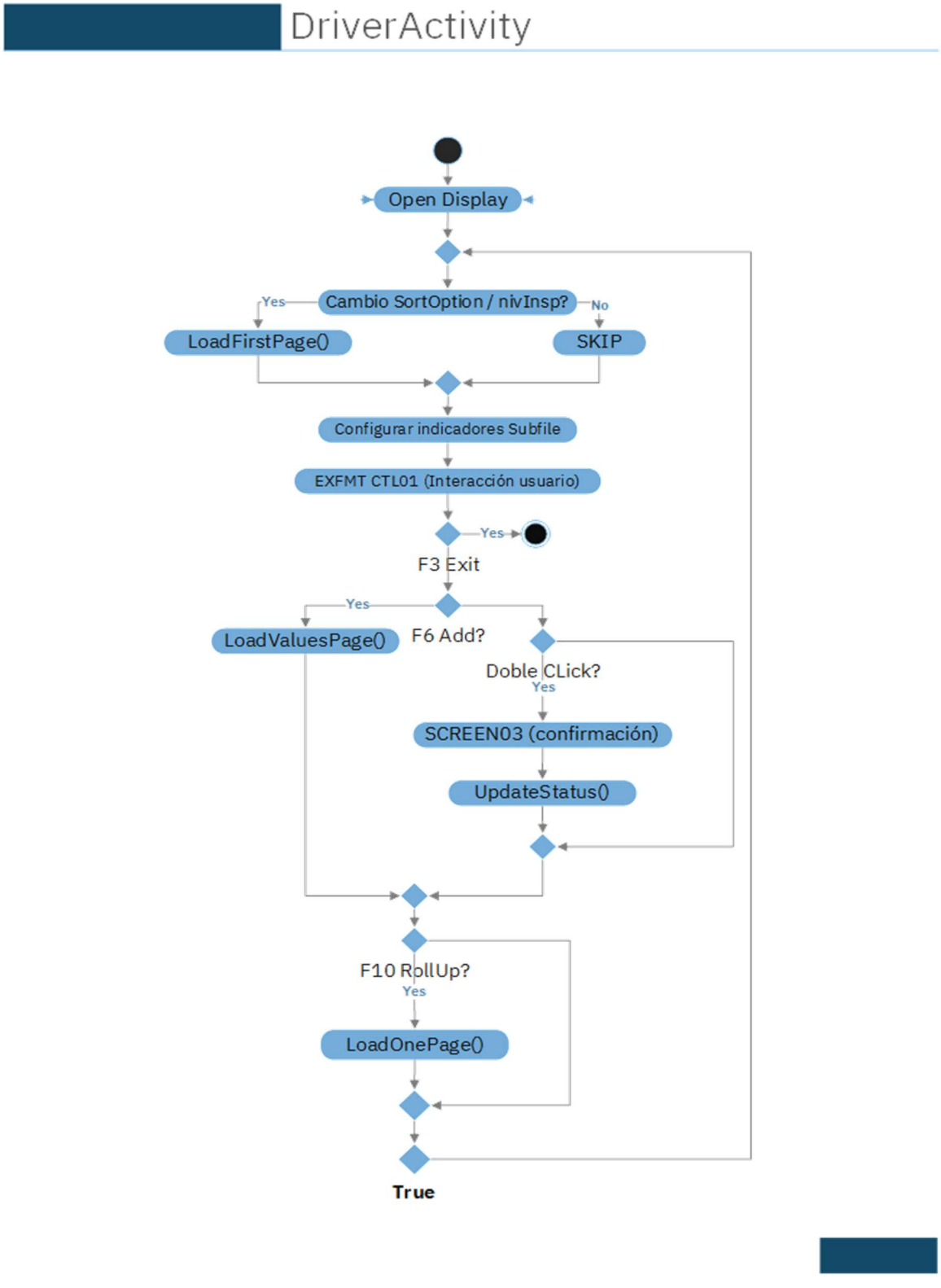


Figura 4.1 -Actividad Procedimiento DRIVE

Resumen conceptual

El procedimiento Driver() convierte el programa RPG en una **máquina de estados controlada por eventos**.

- Cada pantalla es un objeto.
- Cada acción del usuario es un evento.
- El Driver() se asegura de encaminar ese evento hacia la lógica correcta.

En términos de orientación a objetos, Driver() cumple el papel de **controlador principal** (similar a un *Controller* en MVC), donde las pantallas son las **Views** y los registros que manipulan son los **Models**.

4.2 Diagrama global de interacción

El siguiente diagrama integra todos los procedimientos en una vista única, mostrando cómo el flujo principal (*Driver*) coordina con los módulos auxiliares para cargar, navegar y actualizar datos en la aplicación.

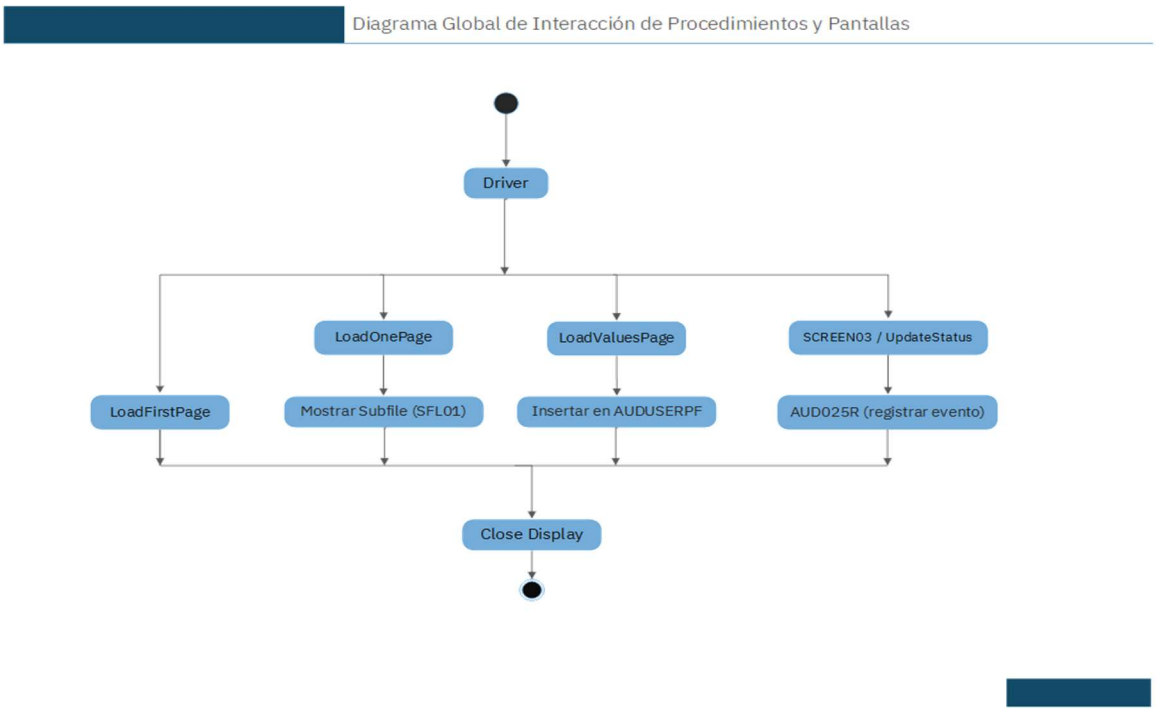


Figura 4.2- Diagrama global de interacción entre procedimientos del sistema RPG OOP.

Capítulo 5. Definición de la Pantalla con SDA/SEU

El diseño de pantallas en IBM i tradicionalmente se realiza con SDA/SEU sobre DDS. En este enfoque, el Display File se concibe como un objeto de presentación con múltiples vistas (record formats), cada una cumpliendo un rol en la interacción con el usuario.

5.1 Fuente completo de la pantalla (DDS)

```
A*%%TS  SD  20250331  215212  JARDILES    REL-V7R5M0  5770-WDS
A                                       DSPSIZ(24 80 *DS3)
A                                       REF(*LIBL/AUDUSERPF)
A                                       INDARA
A                                       CA03(03 'Exit')
A                                       CA06(06 'Add')
A                                       MOUBTN(*ULD ENTER)
A          R SFL01                      SFL
A  38                                   SFLNXTCHG
A            USERAUD   R        O  7  1
A  39                                   COLOR(RED)
A            FNAME     R        O  7 13
A            TYPUSER   R        O  7 59
A            LNAME     R        O  7 34
A            STATUS    R        O  7 57
A  39                                   DSPATR(RI)
A  39                                   COLOR(RED)
A          R CTL01                      SFLCTL(SFL01)
A                                       SFLSIZ(0006)
A                                       SFLPAG(0005)
A                                       WINDOW(2 2 13 70 *NOMSGLIN)
A                                       ENTFLDATR
A                                       RTNCSRLOC(&REC &FLD)
A                                       ROLLUP(10)
A                                       RTNDTA
A                                       SFLCSRRRN(&C1RRN)
A  42                                   SFLDSP
A  41                                   SFLDSPCTL
A  40                                   SFLCLR
A  43                                   SFLEND(*SCRBAR *MORE)
A                                       WDWTITLE((*TEXT &TITLE) (*COLOR TRQ) (*DSPATR
RI))
A                                       WDWTITLE((*TEXT ' F03= Salir   F06= Agregar ')
(*COLOR WHT) (*DSPATR HI) *LEFT *BOTTOM)
```

```
A            SFLRCDNBR      4S 0H      SFLRCDNBR(*TOP)
A            TOPRECIN       5S 0H      SFLSCROLL
A            C1RRN          5S 0H
A            MODE           1A  H
A            REC           10A  H
A            FLD           10A  H
A            TITLE         60A  P
A                                  2  5'Sort:'
A                                     COLOR(BLU)
A            SORTOPTION     1A  B  2 11
A                                  2 15'1=Usuario'
A                                     COLOR(WHT)
A                                  3 15'2=Funsión'
A                                     COLOR(WHT)
A                                  2 39'Select department:'
A                                     COLOR(BLU)
A            NIVINSP        2A  B  2 58
A                                  5  1' Usuario                  Nombre   Funsión   '
A                                     DSPATR(HI)
A                                     DSPATR(UL)
A                                     COLOR(TRQ)
A                                     DSPATR(RI)
A          R SCREEN01
A            MESSAGE       78A  O 17  2COLOR(RED)
A          R SCREEN02
A                                     WINDOW(2 2 13 70 *NOMSGLIN)
A                                     WDWTITLE((*TEXT &TITLE) (*COLOR TRQ) (*DSPATR
RI))
A                                     CA04(04 'Lista')
A                                     CA12(12 'Atras')
A                                     OVERLAY
A                                     PUTOVR
A                                     RMVWDW
A            QTYNRO    R           O  1 64REFFLD(QTYNRO)
A                                     DSPATR(RI)
A                                     COLOR(WHT)
A            FNAME     R           B  3 40REFFLD(FNAME)
A  50                                 DSPATR(RI)
A  50                                 COLOR(RED)
A  50                                 DSPATR(PC)
A            LNAME     R           B  5 44REFFLD(LNAME)
A  51                                 DSPATR(RI)
A  51                                 DSPATR(PC)
A  51                                 COLOR(RED)
A            USERAUD   R           B  7 29REFFLD(USERAUD)
A  52                                 DSPATR(PC)
```

```
A  52                                DSPATR(RI)
A  52                                COLOR(RED)
A                                    CHECK(ER)
A            TYPUSER       10A  H
A            DEPARTMENT    20A  H
A            STATUS         1A  H
A            TITLE         60A  P
A            SLTSU          1Y 0H
A            SLTOP          1Y 0H
A            INDICAP       15A  O  1 48COLOR(WHT)
A                                 3  2'Primer Nombre Inicial Segundo Nombre:'
A                                    DSPATR(HI)
A                                 5  2'Primer Apellido Inicial Segundo Apellido:'
A                                    DSPATR(HI)
A                                 7  2'Usuario para la Auditoria:'
A                                    DSPATR(HI)
A  52                             7 41'<-+-------------------------+'
A                                    COLOR(RED)
A            MSGERROR      40A  B  8 46TEXT('Mensajes de error')
A                                    CNTFLD(020)
A                                    DSPATR(PR)
A                                    COLOR(RED)
A                                 9  1'Tipo de Usuario'
A                                    DSPATR(HI)
A            TYPUSEB        2Y 0B  8 18SNGCHCFLD(*RSTCSR *AUTOENT (*NUMCOL 1))
A                                    CHOICE(1 '>Supervisor' *SPACEB)
A                                    CHCCTL(1 &SLTSU)
A                                    CHOICE(2 '>Operador' *SPACEB)
A                                    CHCCTL(2 &SLTOP)
A          R SCREEN03
A                                    WINDOW(8 40 5 31 *NOMSGLIN)
A                                    CA11(11 'Confirma')
A                                    CA12(12 'Atras')
A                                    TEXT('Mensajes del Proceso')
A                                    OVERLAY
A                                    WDWBORDER((*COLOR RED))
A  69                             5 12'Enter = continue.'
A                                    COLOR(TRQ)
A  69                             2  1'Error General en datos locales'
A                                    DSPATR(HI)
A N69                             2  3'Confirme Desactivación'
A                                    COLOR(WHT)
A N69                             3  3'del Ususrio:'
A                                    COLOR(WHT)
A N69        USERAUD   R        O  3 16REFFLD(USERAUD)
A                                    DSPATR(RI)
A                                    COLOR(RED)
```

5.2 Análisis de componentes

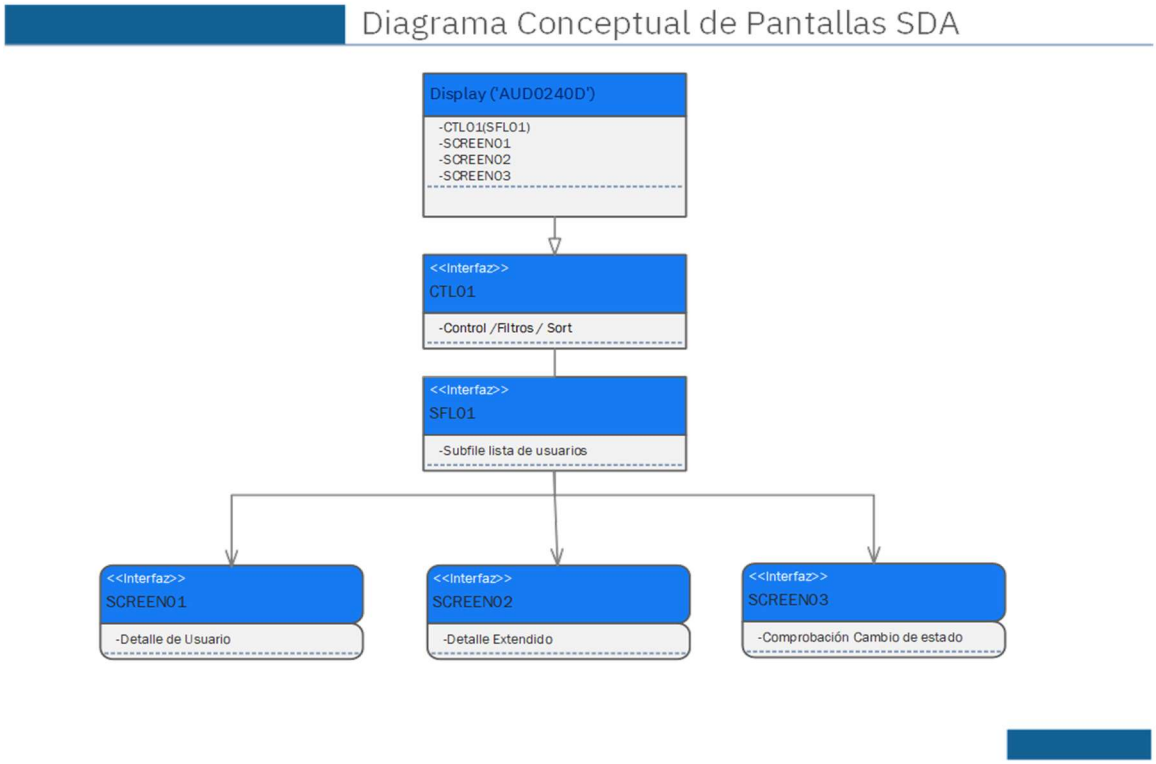


Figura 5.1-Concepto de Pantallas SDA

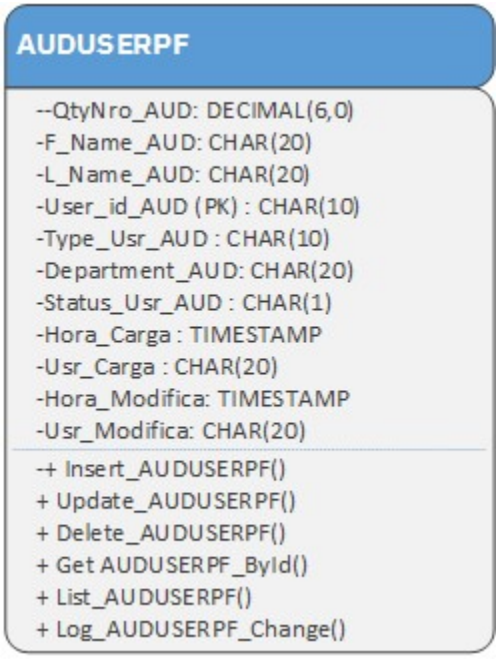


Figura 5.2 -Diagrama UML de la entidad AUDUSERPF, mostrando sus atributos principales y las operaciones CRUD asociadas.

- **Subfile SFL01 y CTL01**

  Permiten mostrar la lista de usuarios en forma paginada con scroll y barra de más registros.

  - SFLDSP, SFLDSPCTL, SFLCLR, SFLEND → control de despliegue.
  - ROLLUP(10) → habilita avance de página.
  - RTNCSRLOC y SFLCSRRRN → permiten capturar posición del cursor y registro seleccionado.

- **SCREEN01**
  Pantalla simple de mensajes (ej. errores de SQL).
- **SCREEN02**
  Ventana de alta de registros con validaciones y mensajes de error. Incluye campos editables (FNAME, LNAME, USERAUD) y controles de formato (DSPATR, COLOR).

- **SCREEN03**
  Ventana emergente de confirmación para operaciones críticas, como desactivar un usuario.

5.3 Conexión con RPG

En el código RPG:

- Cada record format (SFL01, CTL01, SCREEN01, etc.) se referencia con likerec en estructuras de datos.
- Los indicadores (WsInd) permiten controlar atributos visuales definidos en DDS (ej. rojo, subrayado, overlays).
- El procedimiento Driver() gestiona la interacción, alternando entre pantallas con EXFMT y WRITE.

De esta manera, el Display File se convierte en un verdadero **objeto de presentación**, totalmente integrado a la lógica OOP de RPG

.

Capítulo 6. Procedimiento LoadOnePage

El procedimiento *LoadOnePage* es el encargado de cargar registros desde la base de datos DB2/400 hacia la *subfile* definida en el *Display File*.
En el modelo OOP que venimos aplicando, este procedimiento cumple la función de un **método de servicio**, que prepara y proyecta la información hacia la capa de presentación.

6.1 Código fuente completo (RPG Free)

```
Dcl-proc LoadOnePage;
   Dcl-pi *n;
      ioSize          like(RRN01);
      ioEndOfData     ind;
      ouScreen01_rec  likeds(SCREEN01_rec_t);
   end-pi;
   Dcl-ds  SFL01_rec         likeds(SFL01_rec_t)     inz;

   Dcl-ds  DataArray  qualified    dim(SFL01_PAGESIZE);
      useraud     char  (10);
      fname       char  (20);
      typuser     char  (10);
      lname       char  (20);
      status      char  (01);
   end-ds;
   Dcl-s   Ndx          uns(5);
   if ioEndOfData;
      return;
   endif;
   RRN01 = ioSize;
   exec sql  fetch Inp for :SFL01_PAGESIZE rows into :DataArray;
   if SqlState > c_SQL_EOD;
      ouScreen01_rec.Message = '40: SQL failed with state ' + SQLState + '.';
      return;
   endif;
   ioEndOfData = (SqlState >= c_SQL_EOD);
   for Ndx = 1 to sqler3;
      if DataArray(Ndx).Status = 'D';
         WsInd.Cambio_Reg = *on;
         WsInd.Red = *on;
      Else;
         WsInd.Cambio_Reg = *off;
         WsInd.Red = *off;
      Endif;
      eval-corr  SFL01_rec = DataArray (Ndx);
      RRN01 += 1;
      write  Display.SFL01   SFL01_rec;
   endfor;

   ioSize = RRN01;
end-proc LoadOnePage;
```

6.2 Análisis del procedimiento

- **Parámetros de entrada/salida:**
    - ioSize: controla el número de registros cargados.
    - ioEndOfData: indicador de fin de datos en el cursor SQL.
    - ouScreen01_rec: estructura de salida para mensajes.
- **Uso de DataArray:**

  Almacena temporalmente los registros obtenidos con *FETCH*.
- **La instrucción eval-corr:**

  Permite mapear automáticamente campos con el mismo nombre entre DataArray y SFL01_rec.
- **Integración SQL + DDS:**

  fetch obtiene los datos, eval-corr los asigna al formato de pantalla de manera correlativa y write los proyecta en la *subfile*.
- **Control visual con indicadores:**

  WsInd habilita atributos como color rojo para usuarios desactivados.
- **Loop de carga:**

  Recorre DataArray, aplica eval-corr y escribe cada registro en pantalla.

6.2.1 Formatos de pantalla como entidades

Gracias al uso combinado de **likerec y eval-corr**, los formatos de pantalla se comportan como **entidades** que viajan entre procedimientos.

Esto permite tratarlos como **clases de presentación**, manipuladas por métodos como *LoadOnePage* o *Driver*.

6.3 UML del procedimiento LoadOnePage

El siguiente diagrama resume el flujo interno de *LoadOnePage*:

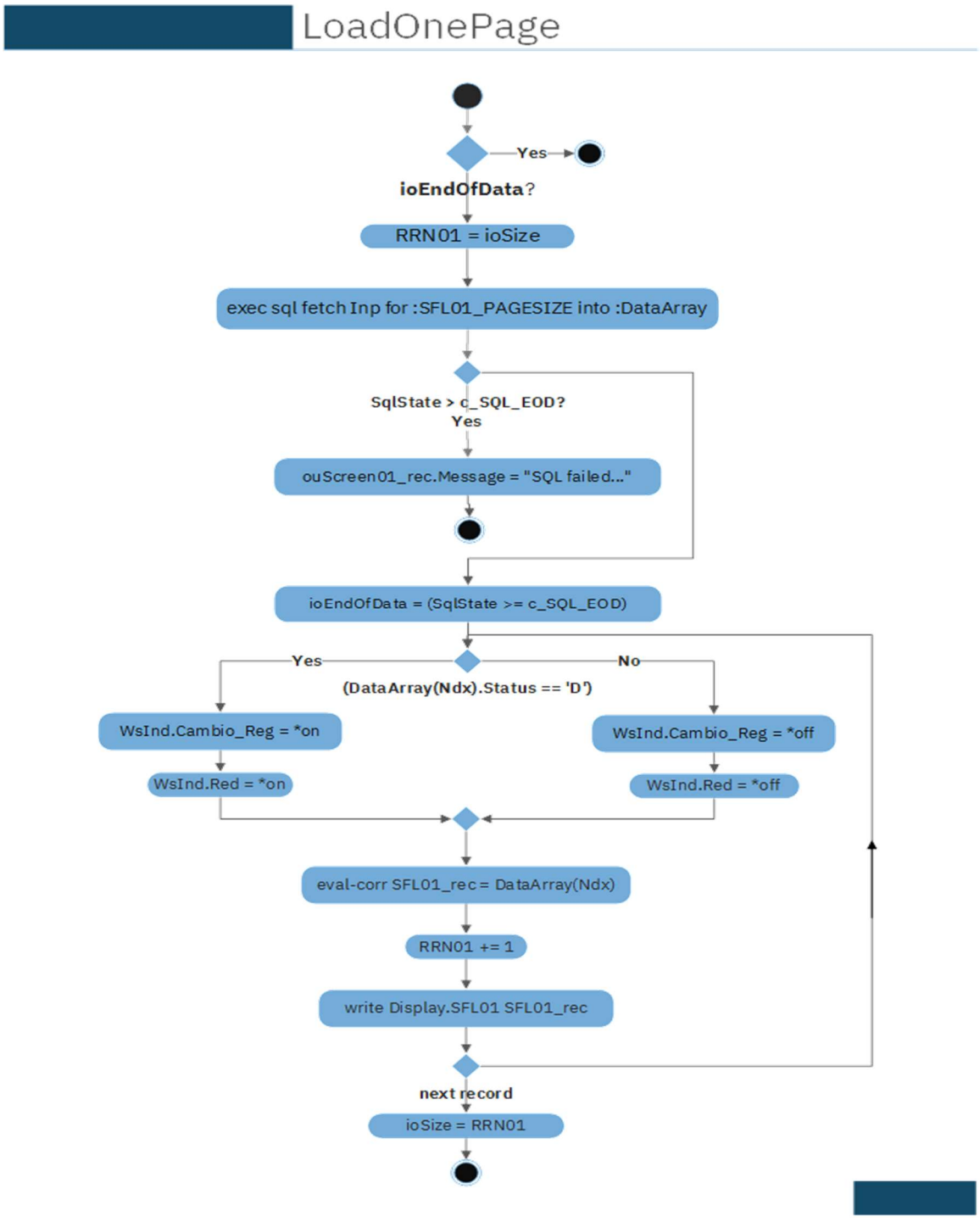


Figura 6.1 – Flujo Procedimiento LoadOnePage

6.4 Comparación con un lenguaje moderno (Java)

```
public void loadOnePage(Connection conn, int pageSize, int offset, List<User>
screenBuffer) throws SQLException {
    String sql = "SELECT useraud, fname, typuser, lname, status FROM AUDUSERPF
LIMIT ? OFFSET ?";
    try (PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setInt(1, pageSize);
        ps.setInt(2, offset);
        try (ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                User u = new User(
                    rs.getString("useraud"),
                    rs.getString("fname"),
                    rs.getString("lname"),
                    rs.getString("typuser"),
                    rs.getString("status")
                );
                screenBuffer.add(u);
            }
        }
    }
}
```

Equivalencia conceptual:

- DataArray en RPG ⇔ List<User> en Java.
- write Display.SFL01 ⇔ agregar a la lista que luego se muestra en GUI.
- ioEndOfData ⇔ validación de fin de resultados con rs.next().

Capítulo 7. Procedimiento LoadFirstPage

El procedimiento ***LoadFirstPage*** tiene como objetivo inicializar la *subfile* con el primer conjunto de registros.
Se ejecuta al comienzo del flujo controlado por el procedimiento *Driver*, y garantiza que la pantalla muestre resultados consistentes según las opciones de ordenamiento (SortOption) y los filtros (Nivinsp) definidos por el usuario.
En términos de diseño orientado a objetos, ***LoadFirstPage*** cumple el rol de un **constructor de la vista**, ya que prepara el contexto de datos para la interacción inicial.

7.1 Código fuente (RPG Free)

```
Dcl-proc LoadFirstPage;

   Dcl-pi *n;
      ioCTL01_rec     likeds(CTL01_rec_t);
      ioSize          like(RRN01);
      ioEndOfData     ind;
      ouScreen01_rec  likeds(SCREEN01_rec_t);
   end-pi;

   Dcl-ds  SFL01_rec    likeds(SFL01_rec_t) inz;

   Dcl-ds  DataArray  qualified dim(SFL01_PAGESIZE);
      useraud   char(10);
      fname     char(20);
      typuser   char(10);
      lname     char(20);
      status    char(01);
   end-ds;

   Dcl-s   Ndx   uns(5);

   // Reinicia indicadores de control
   ioSize = *zero;
   ioEndOfData = *off;

   // Ejecuta SELECT inicial con filtros
   exec sql
      declare Inp cursor for
         select USERAUD, FNAME, TYPUSER, LNAME, STATUS
           from AUDUSERPF
          where NIVINSP like :ioCTL01_rec.Nivinsp || '%'
          order by
             case :ioCTL01_rec.SortOption
                when '1' then USERAUD
                when '2' then TYPUSER
                else USERAUD
             end;
```

```
   exec sql open Inp;

   // Carga primera página
   exec sql fetch Inp for :SFL01_PAGESIZE rows into :DataArray;

   if SqlState > c_SQL_EOD;
      ouScreen01_rec.Message = '40: SQL failed with state ' + SqlState + '.';
      return;
   endif;

   ioEndOfData = (SqlState >= c_SQL_EOD);

   for Ndx = 1 to sqler3;
      eval-corr SFL01_rec = DataArray(Ndx);
      ioSize += 1;
      write Display.SFL01 SFL01_rec;
   endfor;

end-proc LoadFirstPage;
```

7.2 Análisis del procedimiento

- **Inicialización de contexto**

  El procedimiento resetea ioSize y ioEndOfData para garantizar que la subfile se construya desde cero.

- **Uso de filtros dinámicos**

  Los campos SortOption y Nivinsp, provistos por el usuario en la pantalla de control (CTL01), se traducen directamente en cláusulas SQL. Esto permite que la misma rutina sirva para múltiples escenarios de consulta.

- **Similitudes con LoadOnePage**

  - Ambos procedimientos utilizan un *cursor SQL* y el buffer DataArray.
  - Se emplea **eval-corr** para mapear automáticamente los registros.

- **Diferencias principales**
    - *LoadFirstPage* se encarga de **abrir el cursor y declarar el ordenamiento/filtros**.
    - *LoadOnePage* solo avanza sobre el cursor ya abierto.
- **Visión OOP**
  Puede entenderse como el **constructor de la vista inicial**, mientras que *LoadOnePage* actúa como el **método de paginación**.

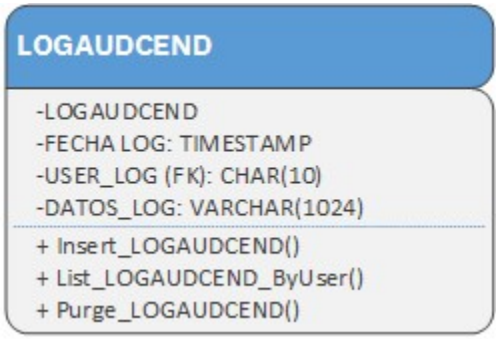


Figura 7.1- Diagrama UML de la entidad LOGAUDCEND, utilizada como tabla de auditoría para registrar eventos de usuario.

7.3 UML simplificado

El flujo de *LoadFirstPage*:

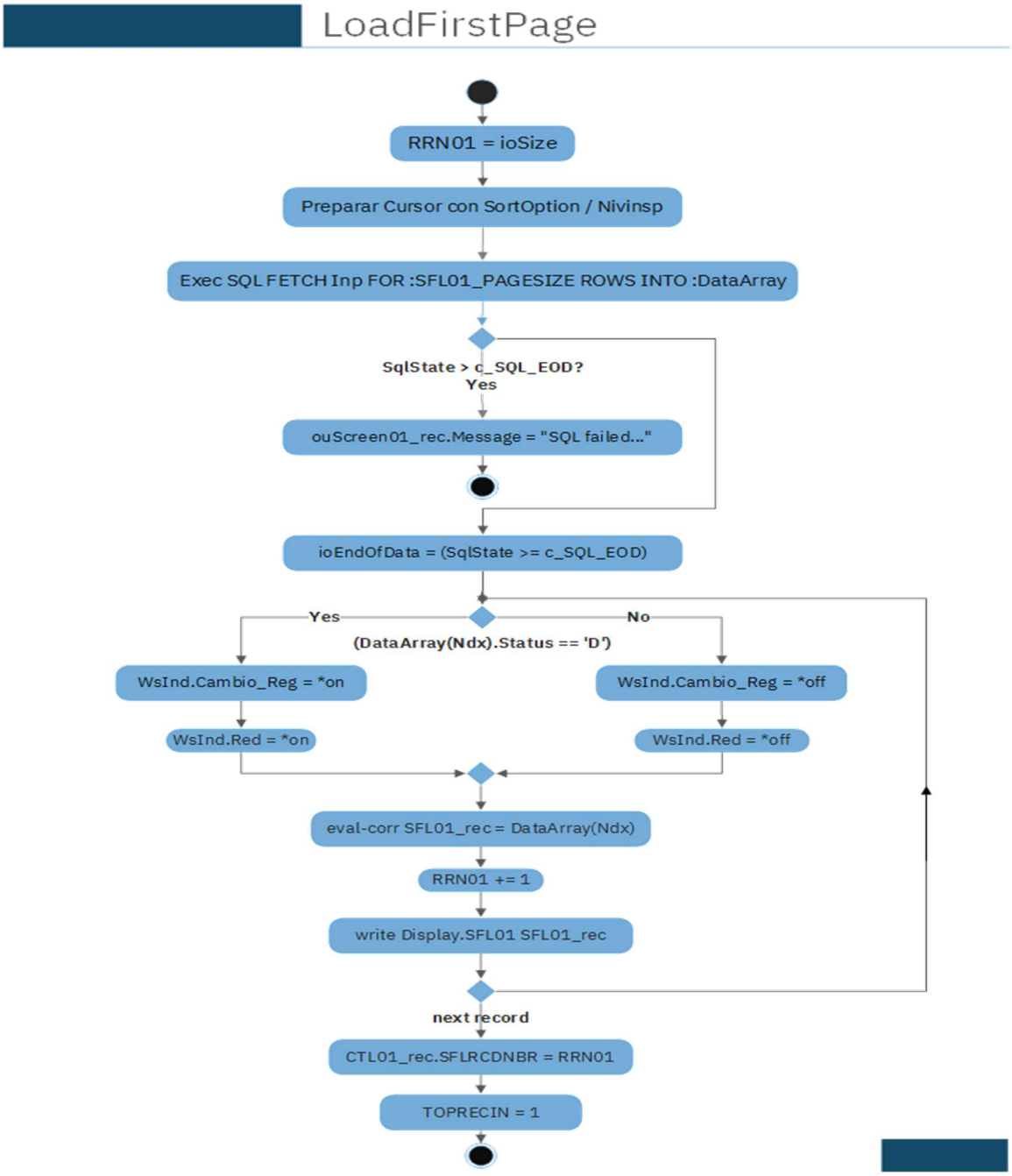


Figura 7.2 – Flujo Procedimiento LoadFirstPage

7.4 Comparación con un lenguaje moderno (Java)

```
public void loadFirstPage(Connection conn, String nivinsp, String sortOption,
int pageSize, List<User> screenBuffer) throws SQLException {
    String orderBy = "USERAUD";
    if ("2".equals(sortOption)) {
        orderBy = "TYPUSER";
    }

    String sql = "SELECT useraud, fname, typuser, lname, status "
               + "FROM AUDUSERPF "
               + "WHERE nivinsp LIKE ? "
               + "ORDER BY " + orderBy + " LIMIT ?";

    try (PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, nivinsp + "%");
        ps.setInt(2, pageSize);
        try (ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                User u = new User(
                    rs.getString("useraud"),
                    rs.getString("fname"),
                    rs.getString("lname"),
                    rs.getString("typuser"),
                    rs.getString("status")
                );
                screenBuffer.add(u);
            }
        }
    }
}
```

Equivalencias conceptuales:

- ioCTL01_rec ⇔ parámetros de entrada nivinsp y sortOption.
- cursor SQL ⇔ PreparedStatement en Java.
- eval-corr ⇔ asignación de atributos al objeto User.
- write Display.SFL01 ⇔ add en la lista screenBuffe

Capítulo 8. Procedimiento LoadValuesPage

El procedimiento *LoadValuesPage* es el encargado de preparar la subfile para la **inserción de un nuevo registro**, cuando el usuario solicita agregar datos (tecla F6).
Se convierte así en el equivalente a un **método constructor** en un lenguaje orientado a objetos: inicializa estructuras, limpia indicadores y establece el contexto para la captura de información.

8.1 Código fuente (RPG Free)

```
Dcl-proc LoadValuesPage;

   Dcl-pi *n;
      ioSize          like(RRN01);
      ioEndOfData     ind;
      ioCTL01_rec     likeds(CTL01_rec_t);
      ioSFL01_rec     likeds(SFL01_rec_t);
   end-pi;

   // Reiniciar campos de la pantalla
   clear ioSFL01_rec;
   clear ioCTL01_rec.SortOption;

   // Preparar indicadores
   WsInd.SflClr    = *on;
   WsInd.SflDspCtl = *on;
   WsInd.SflDsp    = *off;

   // Mostrar pantalla de captura
   exfmt Display.SFL01 ioSFL01_rec;

   if not WsInd.ExitKey;
      // Validar datos obligatorios
      if %len(%trim(ioSFL01_rec.UserAud)) = 0;
         ioCTL01_rec.Message = 'El campo Usuario no puede estar vacío.';
         return;
      endif;

      // Insertar en tabla
      exec sql
         insert into AUDUSERPF (USERAUD, FNAME, LNAME, TYPUSER, STATUS)
         values(:ioSFL01_rec.UserAud, :ioSFL01_rec.Fname,
                :ioSFL01_rec.Lname, :ioSFL01_rec.TypUser,
                :ioSFL01_rec.Status);
```

```
      if SqlState <> '00000';
         ioCTL01_rec.Message = 'Error al insertar registro. SQLSTATE=' +
SqlState;
         return;
      endif;

      // Actualizar subfile
      ioSize += 1;
      write Display.SFL01 ioSFL01_rec;
   endif;

end-proc LoadValuesPage;
```

8.2 Análisis del procedimiento

- **Inicialización**
  Limpia los registros de control y prepara la pantalla para ingreso de un nuevo dato.

- **Manejo de indicadores**
  Se controlan indicadores como SflClr, SflDspCtl y SflDsp para asegurar que la subfile quede lista para aceptar un nuevo registro.

- **Validación**
  Antes de insertar, se verifica que los campos obligatorios no estén vacíos (UserAud en este caso).

- **Inserción SQL**
  Se utiliza un INSERT INTO sobre la tabla AUDUSERPF, aplicando directamente los valores ingresados por el usuario en la pantalla.

- **Actualización de la subfile**
  Una vez insertado, el registro también se escribe en la *subfile*, manteniendo sincronía entre base de datos y pantalla.

- **Diferencias con otros procedimientos**
  - ***LoadFirstPage* y *LoadOnePage*** leen datos existentes.
  - ***LoadValuesPage*** crea un registro nuevo.

8.3 UML simplificado

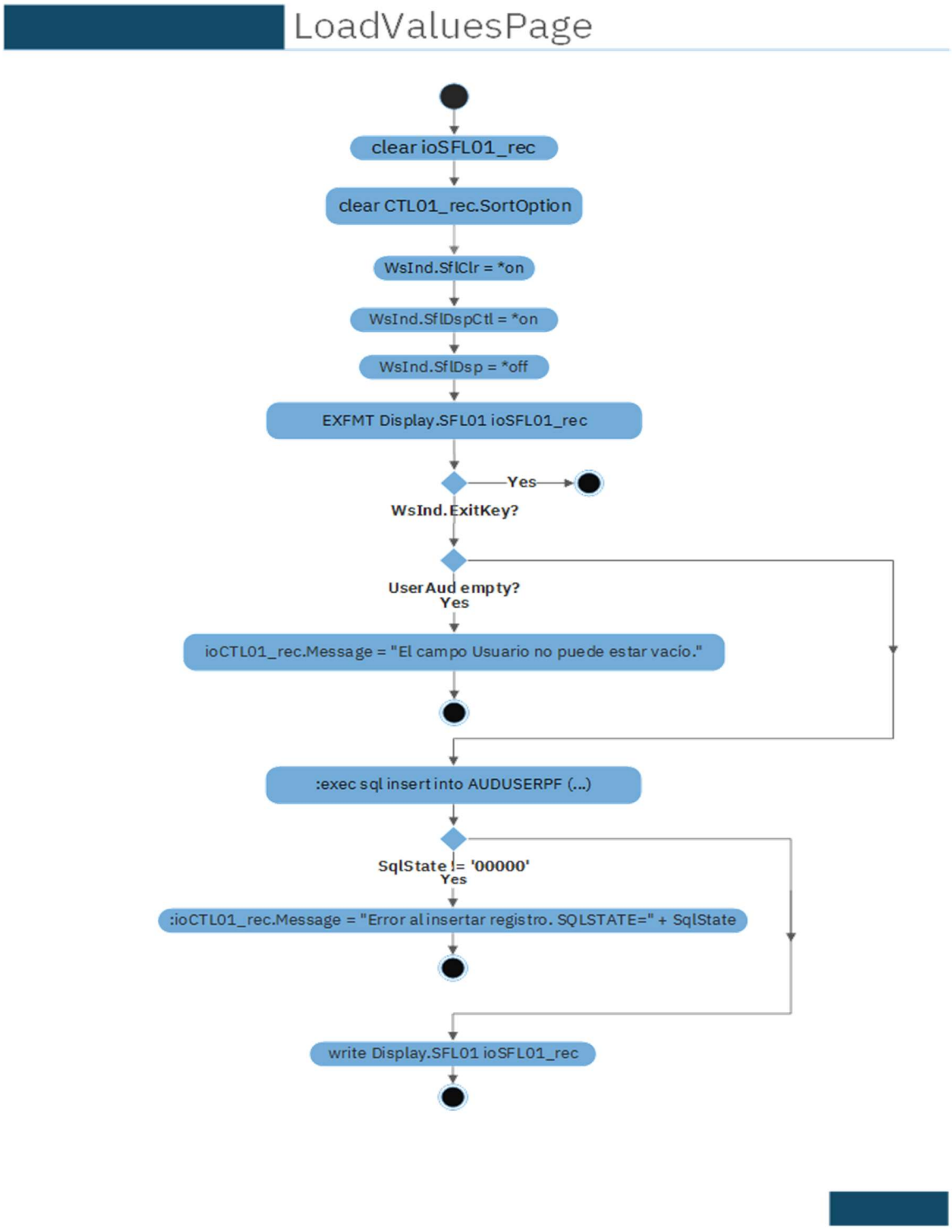


Figura 8.1– Flujo del procedimiento LoadValuesPage.

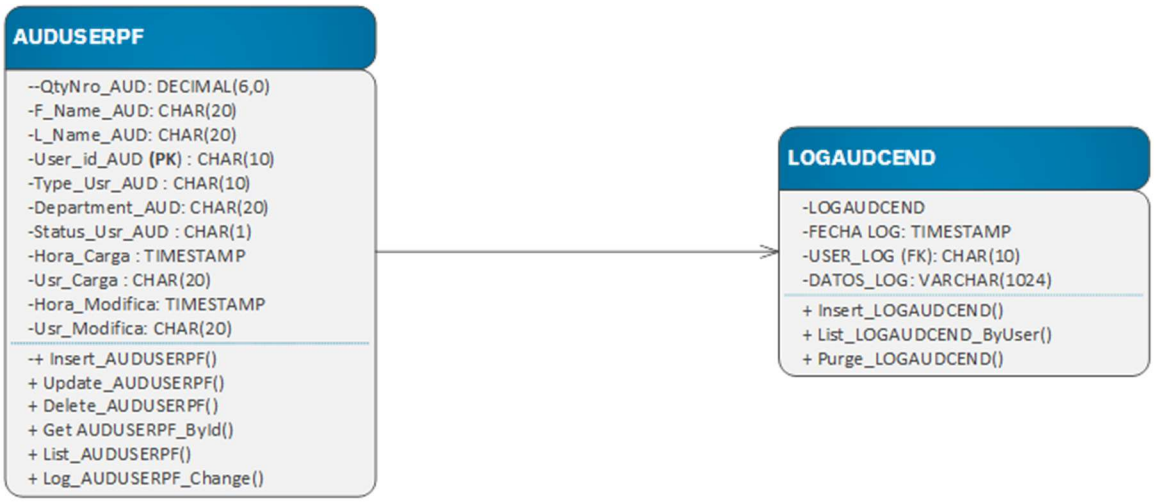


Figura 8.2 -Diagrama UML combinado de AUDUSERPF y LOGAUDCEND, con la relación entre ambas y los procedimientos que las afectan

8.4 Comparación con un lenguaje moderno (Java)

```
public void loadValuesPage(Connection conn, User newUser, List<User>
screenBuffer) throws SQLException {
    if (newUser.getUserAud() == null || newUser.getUserAud().isEmpty()) {
        throw new IllegalArgumentException("El campo Usuario no puede estar
vacío");
    }
    String sql = "INSERT INTO AUDUSERPF (useraud, fname, lname, typuser,
status) VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, newUser.getUserAud());
        ps.setString(2, newUser.getFname());
        ps.setString(3, newUser.getLname());
        ps.setString(4, newUser.getTypuser());
        ps.setString(5, newUser.getStatus());
        ps.executeUpdate();
    }

    // Actualizar la vista (subfile)
    screenBuffer.add(newUser);
}
```

Equivalencias conceptuales:

- ioSFL01_rec ⇔ objeto User en Java.
- INSERT INTO AUDUSERPF ⇔ PreparedStatement.executeUpdate().
- write Display.SFL01 ⇔ screenBuffer.add(newUser).

Capítulo 9 – Procedimiento de cambio de estado (SCREEN03)

El procedimiento asociado a SCREEN03 controla el flujo de cambio de estado de los usuarios en la aplicación. Se ejecuta cuando, desde la subfile principal, el operador realiza doble clic o selecciona un registro con estado activo/inactivo, solicitando su modificación. En términos de orientación a objetos, este procedimiento representa un método de actualización de entidad, pues modifica un atributo (status) del objeto User.

9.1 Código fuente (RPG Free)

```
// Fragmento dentro del procedimiento Driver:

if CTL01_rec.C1RRN > *zeros;          // Doble clic sobre registro
   chain CTL01_rec.C1RRN Display.SFL01 SFL01_rec;
   if %found;
      SCREEN03_rec.USERAUD = SFL01_rec.USERAUD;
      DsTrama.DescEven = '*CambioEstatus-' + SFL01_rec;
      ioUserAud = SFL01_rec.USERAUD;

      exfmt Display.SCREEN03 SCREEN03_rec;

      if WsInd.Confirma;              // Confirmación en pantalla
         select;
            when SFL01_rec.STATUS = 'A';
               ioUserStat = 'D';
               SFL01_rec.STATUS = ioUserStat;
            when SFL01_rec.STATUS = 'D';
               ioUserStat = 'A';
               SFL01_rec.STATUS = ioUserStat;
         endsl;

         exec sql
            update AUDUSERPF A
               set STATUS = :ioUserStat
             where A.USERAUD = :ioUserAud;
```

```
        if SqlStt = '00000';
           DsTrama.DescEven = %trim(DsTrama.DescEven) + '> *CambioEstatus-' +
SFL01_rec;
           AUD025R(DsTrama);        // Log del evento
        endif;

        clear CTL01_rec.SortOption;
     endif;
  endif;
endif;
```

9.2 Análisis del procedimiento

- **Activación desde Driver**

  El flujo inicia cuando el usuario selecciona un registro en la subfile (C1RRN), lo que activa la carga de datos en SFL01_rec.

- **Pantalla de confirmación (SCREEN03)**
  Se presenta al operador con los datos del usuario seleccionado y se espera una confirmación explícita antes de modificar el estado.

- **Cambio de estado**
  El atributo STATUS del usuario alterna entre:
  - 'A' → Activo
  - 'D' → Desactivado

- **Persistencia en base de datos**
  Se ejecuta un UPDATE sobre la tabla AUDUSERPF para reflejar el cambio.
- **Registro de eventos**
  Se invoca al procedimiento AUD025R para dejar traza del cambio en el log de auditoría.

9.3 UML simplificado

El flujo de cambio de estado puede representarse así:

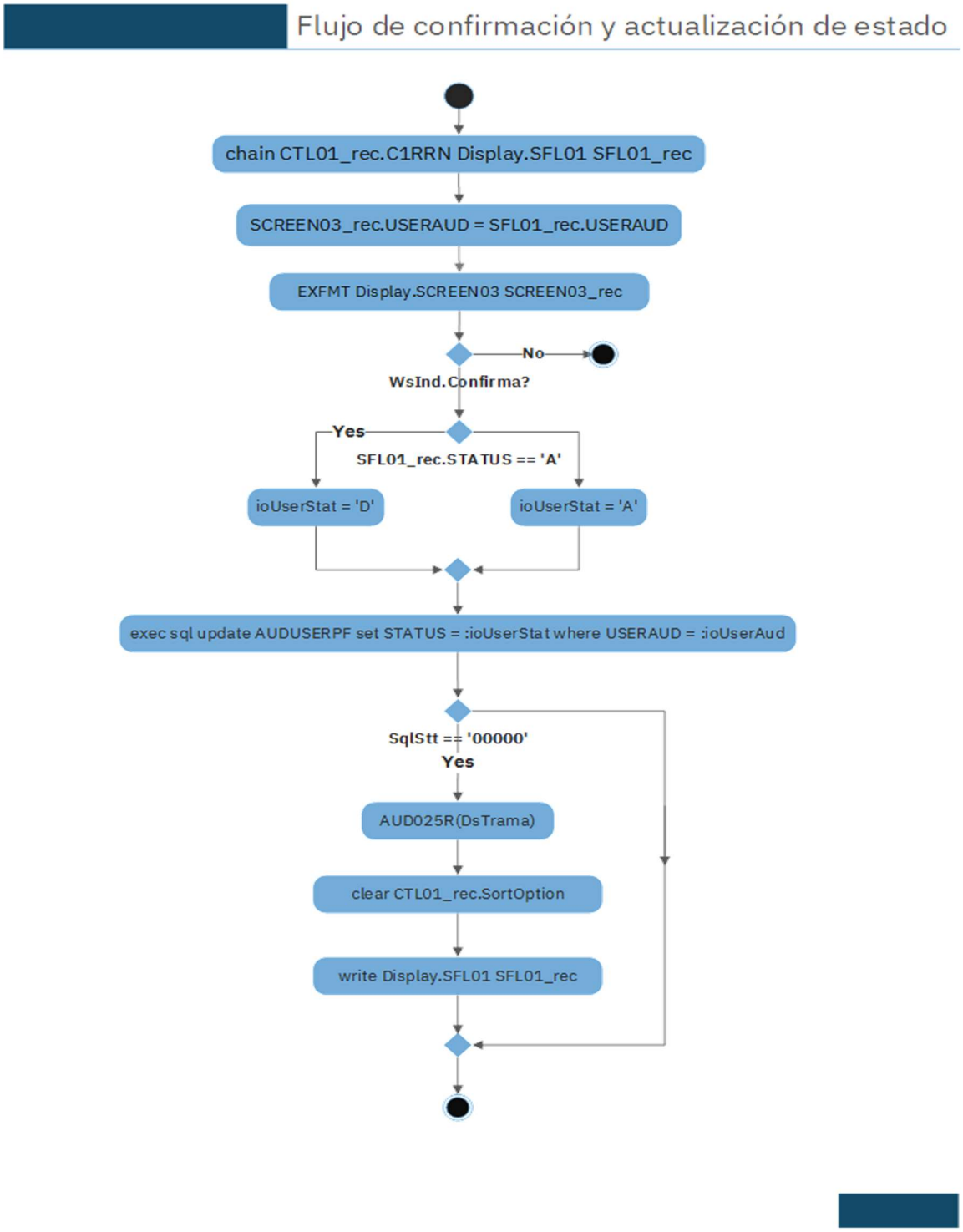


Figura 9.1 - Flujo del cambio de Estado

9.4 Comparación con un lenguaje moderno (Java)

```
public void toggleUserStatus(Connection conn, String userAud) throws
SQLException {
    // Obtener estado actual
    String sqlSelect = "SELECT status FROM AUDUSERPF WHERE useraud = ?";
    String status;
    try (PreparedStatement ps = conn.prepareStatement(sqlSelect)) {
        ps.setString(1, userAud);
        try (ResultSet rs = ps.executeQuery()) {
            if (!rs.next()) return;
            status = rs.getString("status");
        }
    }

    // Alternar estado
    String newStatus = "A".equals(status) ? "D" : "A";

    // Confirmación simulada
    boolean confirmed = true;
    if (confirmed) {
        String sqlUpdate = "UPDATE AUDUSERPF SET status = ? WHERE useraud =
?";
        try (PreparedStatement ps = conn.prepareStatement(sqlUpdate)) {
            ps.setString(1, newStatus);
            ps.setString(2, userAud);
            ps.executeUpdate();
        }
    }
}
```

Equivalencias conceptuales:
- SCREEN03 ⇔ ventana de confirmación en GUI moderna.
- SFL01_rec.STATUS ⇔ atributo status del objeto User.
- exec sql update ... ⇔ PreparedStatement.executeUpdate().
- AUD025R ⇔ servicio de auditoría o logging.

Capítulo 10. Procedimiento de Auditoría (AUD025R)

El procedimiento ***AUD025R*** cumple la función de **registrar eventos de auditoría** en la aplicación.

Toda operación relevante (cambio de estado, inserción de usuario, errores críticos) genera una traza que se almacena en la base de datos o en un archivo de log.

De esta manera se garantiza la **trazabilidad** y se cumplen los principios de control interno y seguridad.

10.1 Código fuente (RPG Free)

```
Dcl-Proc AUD025R;

   Dcl-Pi *n;
      ioTrama likeds(DsTrama) const;
   End-Pi;

   // Insertar en tabla de auditoría
   Exec SQL
      Insert into AUDEVENTPF
         (USEREVEN, PGMEVEN, APPEVEN, DESCEVEN, FECHA, HORA)
      values
         (:ioTrama.UserEven,
          :ioTrama.PgmEven,
          :ioTrama.AppEven,
          :ioTrama.DescEven,
          current_date,
          current_time);

   If SqlState <> '00000';
      // Manejo básico de error
      dsply ('Error en AUD025R. SQLSTATE=' + SqlState);
   EndIf;

End-Proc AUD025R;
```

10.2 Análisis del procedimiento

- **Parámetro de entrada**
  ioTrama es una estructura que encapsula los datos de auditoría: usuario, programa, aplicación y descripción del evento.

- **Persistencia**
  El procedimiento realiza un INSERT en la tabla **AUDEVENTPF**, registrando los metadatos junto con la fecha y hora actuales.

- **Manejo de errores**
  Si ocurre un error SQL, se muestra un mensaje (en producción podría redirigirse a un log de sistema).

- **Uso transversal**
  Este procedimiento se invoca desde:

  - Driver() → en eventos críticos.
  - LoadValuesPage() → al insertar nuevos usuarios.
  - UpdateStatus() → en cambios de estado.

10.3 UML simplificado

AUD025R

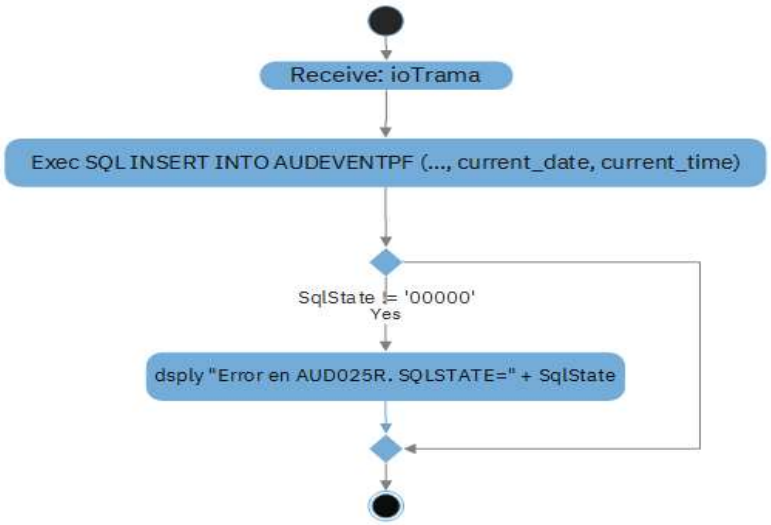


Figura 10.1– Flujo del procedimiento de auditoría AUD025R.

10.4 Comparación con un lenguaje moderno (Java)

```
public void logEvent(Connection conn, AuditEvent event) throws SQLException {
    String sql = "INSERT INTO AUDEVENTPF (usereven, pgmeven, appeven,
desceven, fecha, hora) " +
                 "VALUES (?, ?, ?, ?, CURRENT_DATE, CURRENT_TIME)";
    try (PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, event.getUserEven());
        ps.setString(2, event.getPgmEven());
        ps.setString(3, event.getAppEven());
        ps.setString(4, event.getDescEven());
        ps.executeUpdate();
    }
}
```

Equivalencias conceptuales:

- ioTrama en RPG ⇔ objeto AuditEvent en Java.
- Insert into AUDEVENTPF ⇔ PreparedStatement.executeUpdate().
- dsply error ⇔ logging con Logger.warn/error.

Epílogo – Conclusiones

El recorrido por la modernización del RPG muestra que es posible aplicar técnicas de programación orientada a objetos en entornos tradicionales como AS/400, logrando un puente entre la robustez histórica del sistema y las demandas de la era actual.

La experiencia desarrollada a lo largo de este tutorial ha demostrado que el lenguaje **RPG Free**, tradicionalmente asociado con aplicaciones transaccionales en entornos IBM i (AS/400), puede beneficiarse de un **enfoque orientado a objetos**, aun dentro de sus propias limitaciones.

A través de la construcción paso a paso de un módulo de gestión de usuarios, se evidenció que:

- **La modularidad y encapsulación** mediante procedimientos (*Driver*, *LoadOnePage*, *AUD025R*, etc.) favorece la claridad del código y la reutilización de lógica.
- **El uso de estructuras de datos calificadas** permite modelar pantallas y registros como verdaderas “clases” con atributos, facilitando el trabajo con entidades complejas.
- **La separación de responsabilidades** (pantallas en SDA/SEU, lógica en RPG, persistencia en DB2/400) genera un diseño más mantenible y cercano a las arquitecturas modernas.
- **La comparación con lenguajes actuales como Java** permite visualizar la equivalencia conceptual entre RPG y técnicas OO consolidadas, reduciendo la brecha de aprendizaje para nuevos desarrolladores.

La auditoría transversal (AUD025R) ejemplifica cómo implementar servicios reutilizables, equivalentes a loggers o middlewares en arquitecturas más recientes. .

Más allá del ejemplo técnico, este trabajo también refleja una idea esencial: **la modernización no siempre significa abandonar la tradición**, sino aprender a integrarla con nuevas prácticas. RPG sigue siendo un lenguaje vigente en entornos corporativos críticos, y su evolución hacia estructuras más limpias y modulares lo mantiene útil en la actualidad

En síntesis, este tutorial buscó mostrar cómo, con creatividad y disciplina, es posible **innovar dentro de los límites del propio lenguaje**, preparando el camino para integrar aplicaciones RPG en arquitecturas híbridas y estrategias de modernización digital.

Para finalizar este documento busca ser tanto un tutorial práctico como una reflexión estratégica, mostrando que la innovación no necesariamente rompe con la tradición, sino que puede apoyarse en ella para alcanzar nuevas metas.

 *“Por lo que Innovar desde la tradición hacia la modernidad es posible en el RPG”*

Apéndice 1 – Guía rápida de RPG Free

Este apéndice ofrece una visión práctica de los elementos básicos de **RPG Free**, útil como referencia rápida para el lector.

A.1 Declaraciones y opciones de control

```
Ctl-Opt  Option(*SrcStmt:*NoDebugIO:*NoUnref:*SecLvl)
         DatFmt(*iso) TimFmt(*iso) AlwNull(*usrctl)
         DecEdit('0.') FixNbr(*Zoned) Aut(*All)
         Datedit(*ymd) BndDir('QC2LE')
         Main(MyMainProc);
```

🔹 Configuración general del programa: formato de fechas, compilación, enlace a directorios y procedimiento principal.

A.2 Definición de estructuras de datos

```
Dcl-Ds  WsInd  Len(99)  Qualified;
  ExitKey       ind      Pos(03);
  AddReg        ind      Pos(06);
  RollUp        ind      Pos(10);
  Confirma      ind      Pos(11);
  ErrorNombre   ind      Pos(50);
End-Ds;
```

🔹 Uso de estructuras calificadas para agrupar indicadores de pantalla como atributos de clase.

A.3 Declaración de archivos

```
Dcl-F Display  Workstn(*Ext) Qualified
                 ExtDesc('AUD0240D')
                 ExtFile(*ExtDesc)
                 Sfile(SFL01: RRN01)
                 IndDs(WsInd)
                 UsrOpn;
```

🔷 Asociación entre programa y pantalla física (SDA/SEU).

A.4 Procedimientos y modularidad

```
Dcl-Proc  Driver;
  // Código principal de navegación entre pantallas
End-Proc  Driver;
```

🔷 RPG Free promueve la modularidad dividiendo el flujo en procedimientos independientes.

A.5 Pantallas y subfiles

```
Write   Display.Screen01  Screen01_rec;
Exfmt   Display.CTL01     CTL01_rec;
```

🔷 El subfile permite trabajar con listas dinámicas.
🔷 Exfmt combina escritura y lectura en un solo paso.

A.6 Uso de SQL embebido

```
Exec SQL
   Update AUDUSERPF A
      Set STATUS = :ioUserStat
    Where A.USERAUD = :ioUserAud;
```

🔷 Integración nativa de SQL dentro del flujo RPG.

A.7 Uso de EVAL-CORR

```
For Ndx = 1 to SQLER3;
   Eval-Corr SFL01_rec = DataArray(Ndx);
   RRN01 += 1;
   Write Display.SFL01 SFL01_rec;
EndFor;
```

🔷 **Qué hace**: copia automáticamente los valores de todos los campos con el mismo nombre entre dos estructuras.

🔷 **Ventaja**: reduce la codificación manual de asignaciones campo a campo.

🔷 **Aplicación en el tutorial**: permite manejar los registros del subfile como *entidades*, integrándolos entre los procedimientos (LoadOnePage, LoadFirstPage, etc.) como si fueran objetos transferidos.

A.9 Botones de selección dinámicos en SDA

En **SDA (Screen Design Aid)** se pueden definir **campos de selección dinámica** (choice fields), que actúan como botones en la pantalla.

El ejemplo típico es la definición con SNGCHCFLD, que permite mostrar opciones y asignarles valores variables según la lógica del programa RPG.

```
A          R SCREEN04
A                                  WINDOW(10 5 4 18 *NOMSGLIN)
A                                  OVERLAY
A                                  WDWBORDER((*COLOR RED))
A            FLD004    2Y 0B  1  2SNGCHCFLD(*RSTCSR (*NUMCOL 1) *AUTOENT)
A                                  CHOICE(1 '>Supervisor' *SPACEB)
A                                  CHCCTL(1 &SLTSUP4)
A                                  CHOICE(2 '>Operador' *SPACEB)
A                                  CHCCTL(2 &SLTOPR4)
A            SLTSUP4   1Y 0H
A            SLTOPR4   1Y 0H
```

🔷 **Explicación**:

- SNGCHCFLD: define un campo de selección simple (*single choice field*).
- CHOICE(n 'texto'): define la opción que verá el usuario (ej. *Supervisor*, *Operador*).
- CHCCTL(n &var): vincula la opción con una variable o indicador, permitiendo controlar o leer qué botón se seleccionó.

🔷 **Cómo dar valores variables**:
En el RPG, esas variables (&SLTSUP4, &SLTOPR4) pueden ser manipuladas dinámicamente, de modo que el botón aparezca marcado, activo o inactivo según la lógica de negocio.

Esto refuerza el concepto de que **los botones de selección no son estáticos**: pueden estar controlados como atributos de clase en el RPG, lo que potencia la idea de orientación a objetos aplicada a las pantallas SDA.

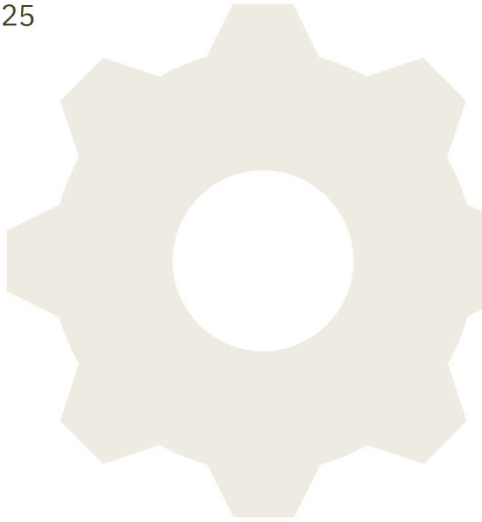
A.10 Conclusión

El **RPG Free** combina la potencia del legado AS/400 con estructuras modernas. Este tutorial mostró cómo conceptos de **orientación a objetos** pueden integrarse incluso en este lenguaje legendario.

Autor: Jesús Ardiles
Versión: 1.0 – Octubre 2025

Apéndice 2 – Anexo técnico de plantillas

ARAF-OOP Templates · Volumen 1
Arquitectura para la Modernización en IBM i
Autor: Jesús Ardiles
Versión: 1.0 – Octubre 2025



ARAF-OOP Templates

· Volumen 1

Modelo Ardiles — Arquitectura para la Modernización en IBM i

Autor: Jesús Ardiles
Versión: 1.0 – Octubre 2025

Introducción

El presente volumen reúne las **plantillas estructurales del modelo ARAF-OOP**, un marco metodológico que moderniza aplicaciones **RPG IV / RPG Free** sobre IBM i mediante prácticas modulares y orientadas a objetos.

Las plantillas definen estándares reutilizables para lograr tres objetivos esenciales:

- Consistencia entre módulos.
- Reducción del consumo de memoria y CPU (eficiencia del hardware).
- Integración nativa con SQL, SDA y componentes de auditoría.

El uso del modificador **TEMPLATE** en variables comunes evita reservar memoria innecesaria, permitiendo que solo las variables efectivamente instanciadas ocupen espacio al ejecutarse.

Esto, junto con la trazabilidad integrada y el diseño desacoplado, constituye el **núcleo técnico del método ARAF-OOP**.

Autor: Jesús Ardiles
Versión: 1.0 – Octubre 2025

Plantilla 1 – Definiciones Comunes (ARAF_CommonDefs.tpl)

Propósito de ARAF_CommonDefs.tpl

Centralizar tipos, constantes y estructuras base que se reutilizan en todos los programas.

Código de ARAF_CommonDefs.tpl

```
** ============================================================
**  ARAF-OOP Template – Definiciones Comunes
** ============================================================
DCL-s TmplStandardSql     Char(1024) Inz(*blanks) Template;
DCL-s TmplStandardFld     Char(10)   Inz(*blanks) Template;
DCL-s TmplSqlStateNoRow   Char(5)    Inz('02000')  Template;
DCL-s TmplMoreRows        Char(5)    Inz('True')   Template;
DCL-s TmplReturnedsql     Char(5)    Inz(*blanks)  Template;
DCL-s TmplMsg70           Char(70)   Inz(*blanks)  Template;
DCL-s TmplRows            Int(10)    Inz(*Zeros)   Template;
DCL-s TmplFldNum          Zoned(9:0) Inz(*Zeros)   Template;
DCL-s TmplFecha8          Zoned(8:0) Inz(*Zeros)   Template;
DCL-s TmplDateSys         Date(*ISO) Inz(*Job)     Template;
DCL-s TmplIND             IND                      Template;
```

Ejemplo de uso

```
DCL-s wFECHACREA  Like(TmplFecha8);
DCL-s WPGAREP     Like(TmplStandardFld);
DCL-s SqlString   Like(TmplStandardSql);
```

Plantilla 2 – Control de Estados SQL (ARAF_SQL_Handle.tpl)

Propósito de ARAF_SQL_Handle.tpl
Normalizar la gestión de estados y errores SQL en todo el sistema.

Código de ARAF_SQL_Handle.tpl

```
DCL-c SQLSTATE_OK     '00000';
DCL-c SQLSTATE_EOD    '02000';
Monitor;
   Exec SQL ... ;
   If SQLSTT = SQLSTATE_OK;
      // Operación exitosa
   ElseIf SQLSTT = SQLSTATE_EOD;
      // Fin de datos
   Else;
      ARAF_AuditWrite('Error SQL: ' + SQLSTT:'SQL':'E');
   EndIf;
EndMon;
```

Plantilla 3 – Formateo de Mensajes (ARAF_MSG_Format.tpl)

Propósito de ARAF_MSG_Format.tpl
Estandarizar mensajes de sistema y usuario para interfaz SDA o logs.

Estructura de ARAF_MSG_Format.tpl

```
DCL-s MsgPrefix Char(10) Inz('[ARAF]') Template;
DCL-s MsgBuffer Char(256) Template;


DCL-Proc ARAF_MsgFormat Export;
  DCL-PI *N Char(256);
    pText Char(200) Const;
  End-PI;
  Return %Trim(MsgPrefix) + ' ' + %Trim(pText);
End-Proc;
```

Plantilla 4 – Auditoría y Registro de Eventos (ARAF_Audit_Log.tpl)

Propósito de ARAF_Audit_Log.tpl
Registrar cada evento relevante del sistema con trazabilidad estándar.

Código base de ARAF_Audit_Log.tpl

```
DCL-Proc ARAF_AuditWrite Export;
  DCL-PI *N;
     pMsgText   Char(256) Const;
     pAuditType Char(10)  Const Options(*NoPass);
     pStatus    Char(1)   Const Options(*NoPass);
  End-PI;
  Exec SQL
```

```
    Insert into LOGDDSDDL
      (FECLOG, USERLOG, DATALOG)
    Values(%Timestamp(*SYS),
           %User,
           '[AUDIT] ' + %Trim(pMsgText));
End-Proc;
```

Buenas prácticas

- Registrar tipo y estado (S éxito / E error).
- Mantener formato [TYPE][STATUS] Mensaje.
- Centralizar auditoría de SQL, UI y Batch.

Plantilla 5 – Ejecución Dinámica SQL (ARAF_SQL_DynamicExec.tpl)

Propósito de ARAF_SQL_DynamicExec.tpl
Permitir ejecución dinámica de sentencias SQL con trazabilidad y auditoría.

Procedimiento base de ARAF_SQL_DynamicExec.tpl

```
DCL-Proc ARAF_SQL_ExecDynamic Export;
  DCL-PI *N;
     pSqlStmt Char(1024) Const;
     pAuditEvent Char(50) Const Options(*NoPass);
  End-PI;
  Exec SQL Prepare DynStmt from :pSqlStmt;
  If SQLSTT = '00000';
     Exec SQL Execute DynStmt;
     ARAF_AuditWrite('SQL ejecutado: ' + %Trim(pSqlStmt),'SQL');
  Else;
     ARAF_AuditWrite('Error SQL: ' + SQLSTT,'SQL':'E');
```

```
  EndIf;
End-Proc;
```

Integración Modular ARAF

Plantilla	Aplicación	Programa	Propósito
CommonDefs	Global	AUD023R, AUD024R	Tipos comunes
SQL_Handle	AUD023R	Control SQL	Excepciones
MSG_Format	AUD024R	Interfaz SDA	Mensajes
Audit_Log	AUD027R	Auditoría global	Trazabilidad
SQL_DynamicExec	AUD023R	Núcleo	SQL dinámico

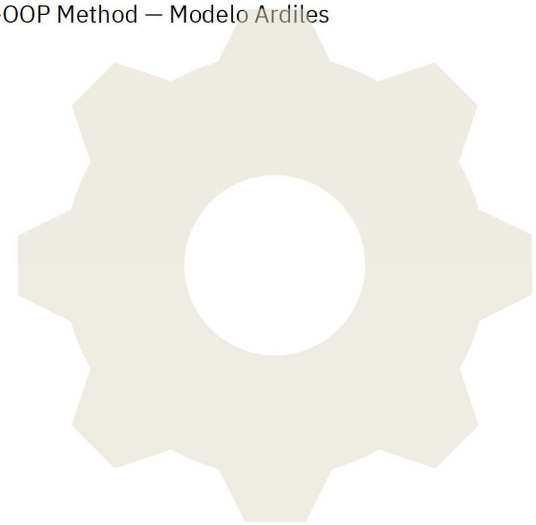
Autor: Jesús Ardiles
Versión: 1.0 – Octubre 2025

Reflexión Final

“En ARAF-OOP, cada plantilla es una pieza del pensamiento arquitectónico. No se trata solo de escribir menos código, sino de escribirlo con propósito.”

Próximas Plantillas – Volumen 2

- **SDA/UI Template:** generación de pantallas modulares y eventos.
- **JSON/REST Template:** exportación de datos e interoperabilidad.
- **API Integration Template:** comunicación entre módulos ARAF y sistemas externos.



“Lo legendario nace del legado — de la solidez de lo que perdura y del valor de quienes modernizan sin romper sus raíces.”

— Jesús Ardiles, Modelo ARAF-OOP

Autor: Jesús Ardiles
Versión: 1.0 – Octubre 2025

Créditos y Derechos

ARAF-OOP Method — Arquitectura para la Modernización en IBM i

Versión 1.0 – Octubre 2025

Autor:

Jesús Ardiles

Arquitecto de Modernización IBM i

Creador del método ARAF-OOP

Arquitectura para la Modernización en IBM i

"Innovar sin borrar lo tradicional: transformar el legado en plataforma viva."