Moderne Workflow-Programmierung mit ABAP® Objects

Handbuch für Entwickler

von
Ulrich Mende

1. Auflage

Moderne Workflow-Programmierung mit ABAP® Objects – Mende

schnell und portofrei erhältlich bei beck-shop.de DIE FACHBUCHHANDLUNG

Thematische Gliederung:

SAP

dpunkt.verlag 2012

Verlag C.H. Beck im Internet:
www.beck.de
ISBN 978 3 86490 013 6

5 Die neue Welt: ABAP Objects im Workflow

5.1 Problemstellung

Workflows laufen über Tage, Wochen, Monate oder sogar Jahre. Sie überstehen Systemdownzeiten, Systemupgrades und sogar Releasewechsel. Und nur Bruchteile dieser Zeit werden zur eigentlichen Codebearbeitung in den Methoden der Einzelschrittaufgaben benutzt. Die übrige Zeit warten die Daten von Workitems (von Workflow-Mustern und von Einzelschrittaufgaben) daher in persistenter Form in sogenannten Containern. Ursprünglich waren Container einfach transparente Tabellen (SWWCONT, SWW_CONTOB), heute wird als Persistenzform die sogenannte XML-Persistenz (Tabelle SWWCNTP0) bevorzugt. Dabei ist es wichtig, dass in diesen Containern nie die Anwendungsdaten selbst, sondern immer nur Referenzen auf die eigentlichen Anwendungsdaten in den Anwendungstabellen gehalten werden. Erst unmittelbar vor der Ausführung eines Workitems wird aus solch einer persistenten Referenz ein Objekt im Hauptspeicher, dessen zugehörigen Instanzmethoden aufgerufen werden können.

Nun ist SAP Business Workflow eine modulübergreifende Querschnittstechnologie. Das Workflow-System muss daher mit den unterschiedlichsten Arten von Business-Objekten umgehen können. Dazu geht es generisch vor. Es verlangt von allen Objekten, die im Workflow bearbeitet werden sollen, dass sie sich an gewisse Konventionen halten. Technisch bedeutet das, dass die Klassen von Workflow-Objekten alle das Interface IF_WORKFLOW implementieren müssen. Nach diesen Ausführungen wird deutlich, dass der Gegenstand dieses Interface nur die Konvertierung von Objektreferenzen von der persistenten in die transiente Form und umgekehrt sein kann.

5.2 IF_WORKFLOW

Dieses Interface macht eine beliebige Klasse »Workflow-tauglich«. Dadurch wird die in den BOR-Objekttypen bereits fest eingebaute Konvertierung zwischen flüchtigen Klassenreferenzen (Hauptspeicher, Aufruf von Methoden) und persistenten Referenzen (Container von Workflows und Schritten) im Kontext von ABAP Objects erreicht.

Erst die Verwendung dieses Interface ermöglicht die Kennzeichnung eines oder mehrerer Instanzattribute als »Key-Attribut«.

Die angesprochene Konvertierung wird ausschließlich dann benötigt, wenn auch Instanzen von Workflow-Klassen verwendet werden sollen, d.h., wenn Instanzmethoden aufgerufen werden sollen. Wenn man also eine Klasse im Workflow benutzen will, die ausschließlich statische Attribute und Methoden verwendet, so braucht man dieses Interface gar nicht auszuprägen (muss es aber aus formellen Gründen doch in die Klasse aufnehmen).

Das Erzeugen bzw. Löschen von Klasseninstanzen hat keinerlei Einfluss auf die DB-Daten der Objektinstanz in den Anwendungstabellen. Zu einem Eintrag in der Kopftabelle eines Business-Objektes (Rechnung, Wechselbeleg, Sperrbeleg usw.) können zu einem Zeitpunkt 0..N (N beliebig) persistente oder transiente Objektinstanzen existieren.

Die SAP Workflow Engine behandelt alle Objekte auf generische Weise. Bei der schrittweisen, interpretativen Abarbeitung eines Workflow-Musters entstehen ständige Kontextwechsel, zu denen die Referenzen auf die zu bearbeitenden Objekte persistent gespeichert bzw. wieder in den Hauptspeicher geladen werden müssen, um die entsprechende Instanz einer ABAP-Klasse zu erzeugen.

Beispiel einer Vertragsbearbeitung im Workflow

Der Workflow gelangt an einen Dialogschritt zu einer Vertragsgenehmigung, der zunächst für einige Stunden oder auch Tage im Eingangskorb des überlasteten Empfängers liegen bleibt. Während dieser Zeit existiert die Vertragsreferenz ausschließlich persistent auf der Datenbank.

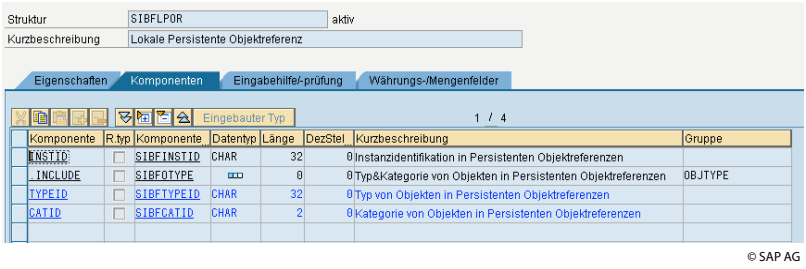
Der Bearbeiter markiert das Workitem (WI) und klickt auf den in der WI-Vorschau angegebenen Link zum Vertrag, um ihn anzuzeigen. In diesem Moment wird automatisch eine Konvertierung der persistenten in die transiente Vertragsreferenz durchgeführt. An der transienten Referenz wird vom Workflow-Laufzeitsystem dynamisch die Displaymethode aufgerufen.

Der Bearbeiter legt das WI zurück, um essen zu gehen → persistente Vertragsreferenz.

Der Bearbeiter führt das Workitem aus, was wieder zu einer Konvertierung persistent → transient führt. Die Methode GENEHMIGEN wird an der so im Hauptspeicher erzeugten Objektreferenz ausgeführt.

Der Schritt ist beendet und die Vertragsreferenz existiert wieder nur auf der DB.

Die transienten Objektreferenzen im Hauptspeicher sind einfach Objektreferenzen nach ABAP Objects, die aus der Klasse und dem Schlüssen in der persistenten Darstellung erzeugt werden. Die persistente Darstellung SIBFLPOR ist eine Struktur aus 3 CHAR-Feldern:

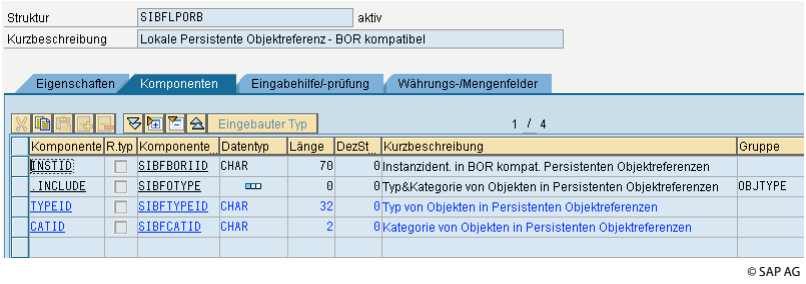

Struktur SIBFLPOR aktiv
Kurzbeschreibung Lokale Persistente Objektreferenz
Eigenschaften | Komponenten | Eingabehilfe/-prüfung | Währungs-/Mengenfelder
Eingebauter Typ 1 / 4

Komponente	R.typ	Komponente	Datentyp	Länge	DezStel	Kurzbeschreibung	Gruppe
INSTID		SIBFINSTID	CHAR	32	0	Instanzidentifikation in Persistenten Objektreferenzen	
.INCLUDE		SIBFOTYPE		0	0	Typ&Kategorie von Objekten in Persistenten Objektreferenzen	OBJTYPE
TYPEID		SIBFTYPEID	CHAR	32	0	Typ von Objekten in Persistenten Objektreferenzen	
CATID		SIBFCATID	CHAR	2	0	Kategorie von Objekten in Persistenten Objektreferenzen	

© SAP AG

Abb. 5–1
Struktur SIBFLPOR für persistente Klassenreferenzen

Während die eingebettete SIBFOTYPE auch für die Typisierung von BOR-Objekten gilt, besitzen diese einen längeren Schlüssel. Deshalb gibt es für BOR-Objekte auch eine andere Persistenzstruktur: SIBFLPORB:


Struktur SIBFLPORB aktiv
Kurzbeschreibung Lokale Persistente Objektreferenz - BOR kompatibel
Eigenschaften | Komponenten | Eingabehilfe/-prüfung | Währungs-/Mengenfelder
Eingebauter Typ 1 / 4

Komponente	R.typ	Komponente	Datentyp	Länge	DezSt	Kurzbeschreibung	Gruppe
INSTID		SIBFBORIID	CHAR	70	0	Instanzident. in BOR kompat. Persistenten Objektreferenzen	
.INCLUDE		SIBFOTYPE		0	0	Typ&Kategorie von Objekten in Persistenten Objektreferenzen	OBJTYPE
TYPEID		SIBFTYPEID	CHAR	32	0	Typ von Objekten in Persistenten Objektreferenzen	
CATID		SIBFCATID	CHAR	2	0	Kategorie von Objekten in Persistenten Objektreferenzen	

© SAP AG

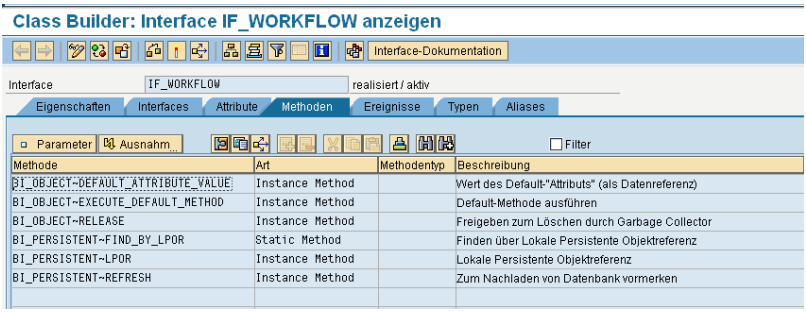
Abb. 5–2
Struktur SIBFLPORB für persistente BOR-Referenzen

Da das generische Laufzeitsystem die einzelnen Workflow-Klassen, mit denen es später arbeiten soll, gar nicht kennen kann, überträgt es die Aufgabe der wechselseitigen Konvertierung an die Klassen selbst. Genau das ist der Gegenstand des Interface BI_PERSISTENT, das das erste eingebettete Teilinterface von IF_WORKFLOW ist.

Außerdem gibt es eine Reihe von Komponenten der SAP Workflow Engine, z.B. das *Workflow-Protokoll*, die Objekte anzeigen können. Hierfür müssen entsprechende Funktionen durch das Objekt bereitgestellt werden.

Das Interface IF_WORKFLOW legt eine logische Klammer um die Interface BI_PERSISTENT (Instanzverwaltung) und BI_OBJECT (Objektverhalten). Das Interface IF_WORKFLOW enthält folgende Methoden:

Abb. 5–3
Interface IF_WORKFLOW im Class Builder (SE24)



© SAP AG

Listing 5–1
Komponenten von BI_OBJECT

```
*"* components of interface BI_OBJECT
interface BI_OBJECT
  public .

 methods DEFAULT_ATTRIBUTE_VALUE
    returning
      value(RESULT) type ref to DATA .
  methods EXECUTE_DEFAULT_METHOD .
  methods RELEASE .
endinterface.
```

Listing 5–2
Komponenten von BI_PERSISTENT

```
*"* components of interface BI_PERSISTENT
interface BI_PERSISTENT
  public .

  Interface BI_OBJECT .

  class-methods FIND_BY_LPOR
    importing
      LPOR type SIBFLPOR
    Returning
      value(RESULT) type ref to BI_PERSISTENT .
  methods LPOR
    returning
      value(RESULT) type SIBFLPOR .
  methods REFRESH .
endinterface.
```

```
*"* components of interface IF_WORKFLOW
interface IF_WORKFLOW
  public .

  Interface BI_PERSISTENT .

  aliases DEFAULT_ATTRIBUTE_VALUE
    for BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE .
  aliases EXECUTE_DEFAULT_METHOD
    for BI_OBJECT~EXECUTE_DEFAULT_METHOD .
  aliases FIND_BY_LPOR
    for BI_PERSISTENT~FIND_BY_LPOR .
  aliases LPOR
    for BI_PERSISTENT~LPOR .
endinterface.
```

Listing 5–3
Komponenten von IF_WORKFLOW

BI_PERSISTANT~FIND_BY_LPOR

Konvertierung Referenz DB → Hauptspeicher

Diese statische Methode wird immer dann gerufen, wenn eine transiente Hauptspeicherinstanz des Objektes benötigt wird, also unmittelbar vor der Ausführung eines Workitems. Mit CREATE OBJECT wird einfach ein neues Objekt vom Typ der implementierenden Klasse angelegt und als RETURNING-Parameter an das Laufzeitsystem zurückgeliefert.

Der RETURNING-Parameter RESULT ist eine Interfacevariable vom Typ Ref To BI_PERSISTENT, der jede Referenz auf eine Klasse zugewiesen werden darf, die das Interface BI_PERSISTENT (oder auch IF_WORKFLOW) unterstützt. Das ist zwangsläufig bei jeder implementierenden Klasse der Fall. Das Workflow-Laufzeitsystem interessiert sich in diesem Moment nicht für die anderen Klassenmethoden der implementierenden Klasse, sondern behandelt alle Klassen bzw. ihre Objekte gleich. Das ist polymorphes Verhalten.

BI_PERSISTANT~LPOR

Konvertierung Referenz Hauptspeicher → DB

Diese Instanzmethode füllt einfach die benötigte persistente Struktur vom Typ SIBFLPOR aus den Instanzdaten und gibt sie zurück.

BI_PERSISTENT~REFRESH

Diese Instanzmethode füllt die Attribute der Instanz mit Werten. Dazu gehören das Lesen von Daten aus den Anwendungstabellen und ggf. der Aufbau von Referenzen auf andere Objekte.

BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE

Diese funktionale Instanzmethode besitzt keine Importparameter, sondern nur einen RETURNING-Parameter. Sie ist vergleichbar den virtuellen Attributen im BOR. Dieser Wert wird im Workflow überall dort verwendet, wo der Schlüssel des Attributes angezeigt werden soll.

BEACHTE: Der Wert wird als Datenreferenz übergeben.

BI_OBJECT~EXECUTE_DEFAULT_METHOD

Diese Instanzmethode ist der Defaultmethode im BOR vergleichbar. Sie hat keine Parameter, sondern kennt nur die Instanzkomponenten. Meist ist es eine Anzeigemethode.

BI_OBJECT~RELEASE

Diese Instanzmethode wird aufgerufen, wenn das Objekt nicht mehr benötigt wird. Letztlich kann der Garbage Collector diese Aufgabe übernehmen.

5.3 Vertragsbearbeitung im Workflow – Klasse ZCL_VERTRAG

Der CONSTRUCTOR merkt sich einfach den Wert des Objektschlüssels in einer Instanzvariablen MV_VERNR und liest den gesamten Satz in die Struktur MS_VERTRAG ein.

Listing 5–4
CONSTRUCTOR der Klasse ZCL_VERTRAG

```
************************************************************
* Parameters:
* IV_VERNR    Importing    TYPE   ZVERNR       Vertragsnummer
************************************************************
method CONSTRUCTOR.
* Schlüsselattribut
  me->mv_vernr =  iv_vernr.

* Vertragsdaten von DB laden
 select single * from ZVERTRAG into MS_VERTRAG
        where vernr = mv_vernr.
endmethod.
```

5.3.1 Attribute MV_VERNR und MS_VERTRAG

Das Attribut MV_VERNR ist der Schlüssel in den Anwendungstabellen und wird daher vorausschauend speziell behandelt.

Für die Attribute gibt es *vor* der Einbeziehung des Interface IF_ WORKFLOW keine Kennzeichnung »Schlüsselattribut«. Dass das Attribut MV_VERNR der Schlüssel der Vertrags-Kopftabelle ist, weiß bis hierher nur der Programmierer, die Klasse weiß es nicht.

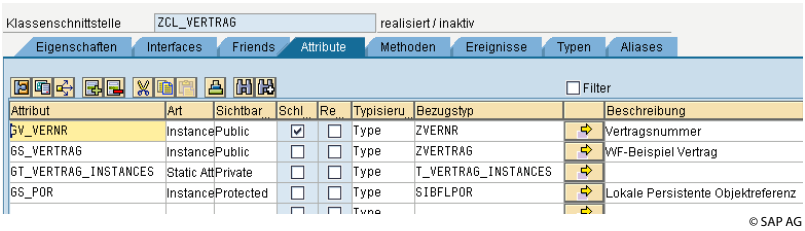


Abb. 5–4
Keine Schlüsselattribute vor IF_WORKFLOW

5.3.2 Methoden ANZEIGEN, AENDERN, GENEHMIGEN

Diese Dialogmethoden (Instanz) rufen direkt den im Kapitel 4 besprochenen FBS Z_VERTRAG jeweils mit der betreffenden Aktion auf. Alle drei Instanzmethoden benutzen das Schlüsselattribut MV_VERNR zur Identifizierung des Vertragsobjektes. Die Methoden ANZEIGEN und AENDERN benötigen überhaupt keine Parameter.

Listing 5–5
Instanzmethode ZCL_VERTRAG.ANZEIGEN

```
**********************************************************************
* Parameters:
*
* Exceptions:
**********************************************************************
method ANZEIGEN.
  CALL FUNCTION 'Z_VERTRAG'
    EXPORTING
      iv_vernr        = MV_VERNR
      iv_aktion       = 'ANZEIGEN'
*   IMPORTING
*     EV_STATUS       =
*     EV_VERNR        =
   EXCEPTIONS
     ABBRUCH         = 1
     OTHERS          = 2
          .
  IF sy-subrc <> 0.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.

endmethod.
```

Die Ausnahme CX_BO_ACTION_CANCELLED wird in der Methode AENDERN geworfen, wenn der User auf *Abbrechen* gedrückt hat. (Das Workflow-System behandelt diese Ausnahme ganz speziell: Es legt ein Workitem, das diese Aufgabe verwendet, zurück in den Eingangskorb.

Listing 5–6
Instanzmethode ZCL_VERTRAG.AENDERN

```
************************************************************
* Parameters:
*
* Exceptions:
* CX_BO_ACTION_CANCELLED   Temporary Bus. Exception: Dialog
************************************************************
METHOD AENDERN.
  CALL FUNCTION 'Z_VERTRAG'
    EXPORTING
      iv_vernr        = MV_VERNR
      iv_aktion       = 'AENDERN'
*   IMPORTING
*     EV_STATUS       =
*     EV_VERNR        =
   EXCEPTIONS
     abbruch         = 1
     OTHERS          = 2.
  IF sy-subrc <> 0.
    RAISE EXCEPTION TYPE cx_bo_action_cancelled.
  ENDIF.
ENDMETHOD.
```

Die Methode GENEHMIGEN hat einen RETURNING-Parameter, der den neuen Status zurückliefert. Im Workflow wird dieser Wert zur Verzweigung benutzt.

Listing 5–7
Instanzmethode ZCL_VERTRAG.GENEHMIGEN

```
************************************************************
* Parameters:
* RV_STATUS  Returning    TYPE  ZSTATUS  Vertragsstatus neu
*
* Exceptions:
* CX_BO_ACTION_CANCELLED    Temporary Bus. Exception: Dialog
************************************************************
METHOD genehmigen.

  DATA: lv_new_status TYPE zstatus.

  CALL FUNCTION 'Z_VERTRAG'
    EXPORTING
      iv_vernr        = mv_vernr
      iv_aktion       = 'GENEHMIGEN'
```

```
    IMPORTING
       ev_status       = lv_new_status
*      EV_VERNR        =
    EXCEPTIONS
      abbruch         = 1
      OTHERS          = 2.

  IF sy-subrc <> 0.
    RAISE EXCEPTION TYPE cx_bo_action_cancelled.
  ELSE.
    rv_status = lv_new_status.
  ENDIF.

ENDMETHOD.
```

Die Methoden selbst prüfen nicht, ob auf der DB tatsächlich ein Vertrag mit der Nummer MV_VERNR existiert. Man geht üblicherweise davon aus, dass ein Vertrag, der auf der DB angelegt wurde und über Änderungsbelege einen Workflow gestartet hat, nicht böswillig von der DB gelöscht wird.

Möchte man allerdings ganz sicher gehen, so muss man in der Methode FIND_BY_LPOR (s. Abschnitt 5.4) stets auf Existenz prüfen, und zwar nicht nur beim Neuanlegen, sondern auch beim Lesen der Objekte aus der Puffertabelle.

5.3.3 Ereignisse CREATED, CHANGED und CANCELLED

Die Ereignisse werden als PUBLIC-Instanzereignisse angelegt. Es werden keine Eventhandlermethoden implementiert, da sich nur der Workflow für das Ereignis interessiert. Das Laufzeitsystem stellt generische Eventhandler für alle Workflow-Klassen bereit:

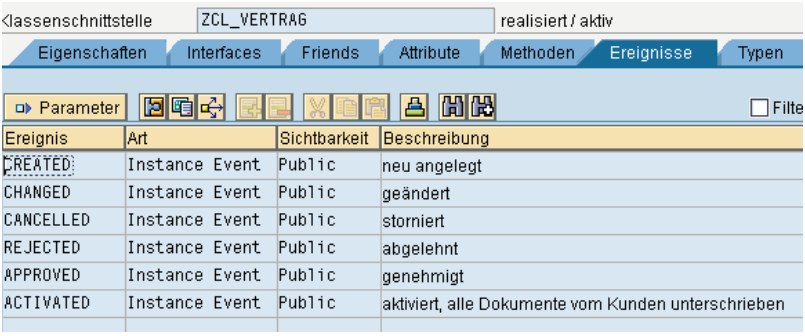


Abb. 5–5
Ereignisse der Klasse ZCL_VERTRAG

5.3.4 Erste Tests der Klasse ohne Interface IF_WORKFLOW

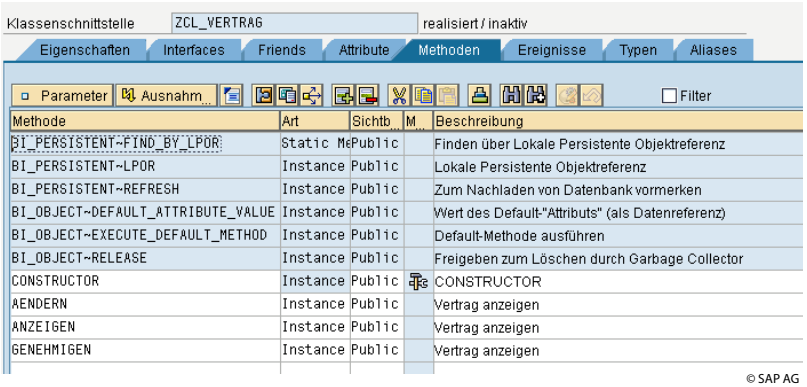
Bereits an dieser Stelle können und sollten alle Komponenten der neuen Klasse getestet werden. Dazu gehören auch die Fehlersituationen wie Abbruch, falsche Vertragsnummer usw.

Je mehr Softwareschichten nämlich dazukommen, desto schwieriger gestalten sich die Tests.

5.4 Einfachste Ausprägung von IF_WORKFLOW

Die Einbindung des IF_WORKFLOW erweitert die Methoden von ZCL_VERTRAG. Die vom Interface »geerbten« und zu implementierenden Methoden sind grau hinterlegt.

Abb. 5–6
Methodenliste von ZCL_VERTRAG nach Einbindung von IF_WORKFLOW



Zunächst lagert man das Nachlesen der Instanzdaten aus dem CONSTRUCTOR in die neue Methode BI_PERSISTENT~REFRESH aus. Sie liest den gesamten Satz der Vertragstabelle in die Objektdaten.

Listing 5–8
Methode BI_PERSISTENT~REFRESH

```
************************************************************
* Parameters:
************************************************************
method BI_PERSISTENT~REFRESH.
  SELECT SINGLE * FROM zvertrag INTO MS_VERTRAG
    WHERE vernr = me->mv_vernr.
  IF sy-subrc <> 0.
*    RAISE EXCEPTION TYPE cx_bo_error.
  ENDIF.
endmethod.
```

Danach wird im CONSTRUCTOR der Aufruf auf BI_PERSISTENT~REFRESH( ) gemacht; so spart man Redundanz.

```
*********************************************************
* Parameters:
* IV_VERNR    Importing  TYPE   ZVERNR     Vertragsnummer
*********************************************************************
method CONSTRUCTOR.
* Schlüsselattribut
  me->mv_vernr =  iv_vernr.

* Vertragsdaten von DB laden
BI_PERSISTENT~REFRESH( ).
endmethod.
```

Listing 5–9
CONSTRUCTOR mit Aufruf auf BI_PERSISTENT~REFRESH

Unter dem Attribute-Reiter ist durch die Einbindung des Interface IF_WORKFLOW nun eine neue Spalte »Schlüssel« zu sehen. Hier markiert man das Schlüsselfeld MV_VERNR. Dieser Schritt ist sehr wichtig. Vergisst man ihn, können Objektinstanzen später ggf. nicht richtig gefunden werden. Insbesondere die TA SWI6 (Workflows zu Objekten) funktioniert ohne Schlüsselkennzeichen gar nicht.

Jeder neue Aufruf von FIND_BY_LPOR erzeugt für denselben Vertrag in der Datenbank ein neues Objekt im Hauptspeicher.

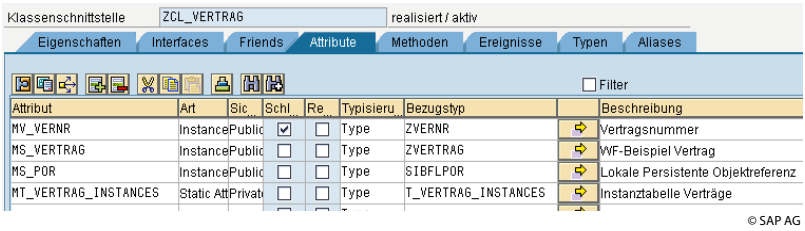


Abb. 5–7
Key-Flag an der Vertragsnummer

Die Klasse wird explizit angegeben, sodass diese Methode bei einer Ableitung redefiniert werden muss.

```
*************************************************************
* Parameters:
* LPOR   Importing  TYPE  SIBFLPOR   Lokale pers. Obj.-Ref
* RESULT  Returning  TYPE REF TO  BI_PERSISTENT  Bus. Ref.
*************************************************************
METHOD bi_persistent~find_by_lpor.
* Konvertierung Container --> Hauptspeicher

  DATA: lo_vertrag          TYPE REF TO zcl_vertrag.
  DATA: lv_vernr            TYPE zvertrag-vernr.

  CHECK lpor-instid IS NOT INITIAL.
```

Listing 5–10
Methode BI_PERSISTENT~FIND_BY_LPOR

```
*- Inst-Id ist die Vertragsnummer
  lv_vernr = lpor-instid.

  TRY.
    CREATE OBJECT lo_vertrag
        EXPORTING
          iv_vernr = lv_vernr.
    CATCH cx_bo_error .
*------ object not found
        EXIT.
  ENDTRY.
* die neu angelegte Instanz wird zurückgegeben
  result = lo_vertrag.
ENDMETHOD.
```

Die Instanzmethode BI_PERSISTANT~LPOR füllt einfach die benötigte persistente Struktur vom Typ SIBFLPOR und gibt sie zurück. Auch hier wird der explizite Klassenname benutzt, was der Vererbung entgegensteht.

Listing 5–11
Methode BI_PERSISTANT~LPOR

```
************************************************************
Parameters:
* RESULT  Returning  TYPE SIBFLPOR   Lokale pers. Obj.-Ref
************************************************************
method BI_PERSISTENT~LPOR.
* Konvertierung Hauptspeicher --> Container

  result-catid  = 'CL'.
  result-typeid = 'ZCL_VERTRAG'.
  result-instid = me->MV_VERNR.
endmethod.
```

Die Instanzmethode BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE schreibt eine Referenz auf die Vertragsnummer der Instanz in das Ergebnis.

Listing 5–12
Methode BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE

```
************************************************************
* Parameters:
* RESULT  Returning  TYPE  Ref To DATA   Ref.auf Datenobjekt
************************************************************
method BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE.
  GET REFERENCE OF MV_VERNR INTO result.
endmethod.
```

Die Instanzmethode BI_OBJECT~EXECUTE_DEFAULT_METHOD ruft hier die Anzeigemethode auf.

```
************************************************************
* Parameters:
************************************************************
method BI_OBJECT~EXECUTE_DEFAULT_METHOD.
  me->anzeigen( ).
endmethod.
```

Listing 5–13
Methode BI_OBJECT~EXECUTE_DEFAULT_METHOD ruft die Anzeigemethode auf.

Die Instanzmethode BI_OBJECT~RELEASE ist leer.

```
method BI_OBJECT~RELEASE.
endmethod.
```

Listing 5–14
Methode BI_OBJECT~RELEASE ist leer.

5.5 Erweiterte Ausprägung IF_WORKFLOW

Hier wird eine erweiterte, deutlich bessere Implementierung vorgestellt, die folgende zusätzliche Leistungen bietet:

- Ermitteln der Objektklasse im CONSTRUCTOR, sodass die Ausprägung von IF_WORKFLOW stabil gegen Vererbung ist und nicht redefiniert werden muss.
- Pufferung angelegter Objektinstanzen in einer statischen Tabelle

Es wird ein Instanzattribut GS_LPOR TYPE SIBFLPOR angelegt, das die persistente Objektreferenz gleich beim Anlegen des Objektes im CONSTRUCTOR speichert. Das ist sinnvoll, weil sich von einem Objekt weder Typ noch Schlüssel ändern können. Weiterhin wird im CONSTRUCTOR anstelle des SELECT SINGLE die vom IF_WORKFLOW verlangte Refreshmethode aufgerufen. Diese kapselt den Datenbankzugriff.

```
************************************************************
Parameters:
* IV_VERNR    Importing    TYPE      ZVERNR    Vertragsnummer
************************************************************
method CONSTRUCTOR.
* Schlüsselattribut
  me->mv_vernr =  iv_vernr.

* Persistente Objektreferenz auf Klasse ZCL_VERTRAG
* oder abgeleitete wird in der Interfacemethode
* BI_PERSISTENT~LPOR zurückgeliefert
  me->gs_por-catid  = 'CL'.
  me->gs_por-typeid = zcl_system=>get_class_of_object( me ).
  me->gs_por-instid = me->mv_vernr.
```

Listing 5–15
CONSTRUCTOR bei erweitertem Interface

```
* Vertragsdaten von DB laden
  me->bi_persistent~refresh( ).

endmethod.
```

Weiterhin wird ein statisches PRIVATE-Attribut zur Pufferung aller Objektinstanzen in einer Tabelle eingeführt. Der Zeilentyp ist T_VERTRAG_INSTANCE, er speichert den Schlüssel (also die Vertragsnummer VERNR) und die zugehörige Objektreferenz nebeneinander. Der darauf aufbauende Tabellentyp ist T_VERTRAG_ INSTANCES.

Listing 5–16
Typen und Klassendaten für Instanzverwaltung

```
*"* private components of class ZCL_VERTRAG
*"* do not include other source files here!!!
private section.

  types:
    BEGIN OF t_vertrag_instance ,
           vernr  TYPE zvernr,
           instance   TYPE REF TO zcl_vertrag,
    END OF t_vertrag_instance .
  types:
    t_vertrag_instances TYPE STANDARD TABLE OF
           t_vertrag_instance .

  class-data MT_VERTRAG_INSTANCES type T_VERTRAG_INSTANCES .
```

Die statische Methode BI_PERSISTANT~FIND_BY_LPOR sucht nun mit dem Schlüssel aus INSTID zunächst in der Puffertabelle, ob es dort bereits eine Referenz mit IV_VERNR gibt. Wenn ja, so wird diese zurückgeliefert. Nur wenn die Referenz nicht vorhanden ist, dann wird sie mit CREATE OBJECT angelegt, in die Puffertabelle eingefügt und zurückgeliefert. Diese Pufferung funktioniert auch bei Vererbung, weil in GT_INSTANCES ja auch Referenzen für abgeleitete Klassen gespeichert werden können (die Schlüsselattribute dürfen sich natürlich bei Ableitung nicht ändern).

Listing 5–17
Methode FIND_BY_LPOR mit Instanzpufferung

```
************************************************************
* Parameters:
* LPOR    I  TYPE        SIBFLPOR      Lokale pers. Obj.-Ref
* RESULT  R TYPE REF TO  BI_PERSISTENT Pers. Bus. Instanz
************************************************************
METHOD bi_persistent~find_by_lpor.
  DA-TA: lo_vertrag          TYPE REF TO zcl_vertrag.
  DATA: lv_vernr            TYPE zvertrag-vernr.
  DATA: ls_vertrag_instance TYPE t_vertrag_instance.
```

```
  CHECK lpor-instid IS NOT INITIAL.

*- Inst-Id ist die Vertragsnummer
  lv_vernr = lpor-instid.

* Die Klassenvariable gt_vertrag_instances verwaltet
* alle Instanzen dieser Klasse, die gleichzeitig im
* Hauptspeicher leben

* Nachsehen, ob es eine Instanz zum Vertrag VERNR gibt
  READ TABLE gt_vertrag_instances WITH KEY vernr = lv_vernr
       INTO ls_vertrag_instance.
  IF sy-subrc <> 0.  " nein, neu anlegen und in die Liste
    TRY.
        CREATE OBJECT lo_vertrag type (lpor-typeid)
          EXPORTING
            iv_vernr = lv_vernr.
      CATCH cx_bo_error .
*------ object not found
        EXIT.
    ENDTRY.
    ls_vertrag_instance-vernr     = lv_vernr.
    ls_vertrag_instance-instance  = lo_vertrag.
    APPEND ls_vertrag_instance TO mt_vertrag_instances.
  ENDIF.

* die gefundene oder neu angelegte Instanz wird zurückgegeben
  result = ls_vertrag_instance-instance.

ENDMETHOD.
```

Die Instanzmethode BI_PERSISTANT~LPOR gibt einfach die bereits gefüllte Struktur GS_POR zurück.

```
method BI_PERSISTENT~LPOR.
  result = me->gs_por.
endmethod.
```

Listing 5–18
Methode LPOR liefert Instanzdatum GS_POR.

Die Instanzmethode BI_PERSISTENT~REFRESH liest den gesamten Satz der Vertragstabelle in den Hauptspeicher.

```
method BI_PERSISTENT~REFRESH.
  SELECT SINGLE * FROM zvertrag INTO MS_VERTRAG
    WHERE vernr = me->MV_VERNR.
  IF sy-subrc <> 0.
*    RAISE EXCEPTION TYPE cx_bo_error.
  ENDIF.
endmethod.
```

Listing 5–19
Methode REFRESH liest Vertragsdaten von der DB

Die Instanzmethode BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE schreibt eine Referenz auf die Vertragsnummer der Instanz in das Ergebnis.

Listing 5–20
Wert des Defaultattributes ist Vertragsnummer.

```
method BI_OBJECT~DEFAULT_ATTRIBUTE_VALUE.
  GET REFERENCE OF MV_VERNR INTO result.
endmethod.
```

Die Instanzmethode BI_OBJECT~EXECUTE_DEFAULT_METHOD ruft hier die Anzeigemethode auf.

Listing 5–21
Defaultmethode ruft parameterlose Anzeigemethode auf.

```
method BI_OBJECT~EXECUTE_DEFAULT_METHOD.
  me->anzeigen( ).
endmethod.
```

Die Instanzmethode BI_OBJECT~RELEASE ist leer.

Listing 5–22
Methode RELEASE bleibt leer.

```
method BI_OBJECT~RELEASE.
endmethod.
```

Wenn man sich ein Gefühl dafür verschaffen möchte, wie häufig die Methode FIND_BY_LPOR aufgerufen wird und wie effektiv die Pufferung der Objektinstanzen ist, dann kann man in diese Methode einen Logpoint-Eintrag zu einer CHECK-POINT-Gruppe schreiben (siehe Kap. 10).

Instanzpufferung: Performance vs. Datensicherheit

Der Grundgedanke der Instanzverwaltung (Instanzpufferung) in einer statischen Liste der Klasse ist die Erhöhung der Performance. In diesem Sinne wird man im Konstruktor möglichst viele (alle?) Daten der Objektinstanz von der Datenbank in entsprechende Instanzattribute einlesen. Bei nachfolgenden Zugriffen auf die Instanz sind die Daten dann bereits vorhanden. Diesem Vorteil stehen zwei potenzielle Nachteile gegenüber:

Redundanz zwischen Objekt- und DB-Daten

- Unbefugte Instanziierung der Klasse an der Instanzverwaltung vorbei
- Datenunterschied zwischen Datenbank und Hauptspeicher

Beide Nachteile können genau dann auftreten, wenn die Anwendungsdaten einer Objektinstanz auf der Datenbank geändert werden, was ja der eigentliche Zweck der gesamten EDV ist. Das unbefugte Instanziieren umgeht man dadurch, dass die Instanzerzeugung einer Klasse auf PRIVATE gesetzt wird. Damit ist sie nur den Methoden der eigenen

Klasse erlaubt. Diese müssen zwingend zur Instanziierung die statische Methode benutzen, die die Instanzverwaltung unterstützt:

```
************************************************************
* Parameters:
* IV_VERNR    I TYPE         ZVERNR       Vertragsnummer
* RV_VERTRAG  R TYPE REF TO  ZCL_VERTRAG  Vertrag
*
* Exceptions:
************************************************************
method GET_REF.
  data: LS_POR type sibflpor.
  ls_por-catid  = 'CL'.
  ls_por-typeid = 'ZCL_VERTRAG'.
  ls_por-instid = iv_vernr.
  rv_vertrag ?= find_by_lpor( ls_por ).
endmethod.
```

Listing 5–23
Statische Methode zur Instanzerzeugung

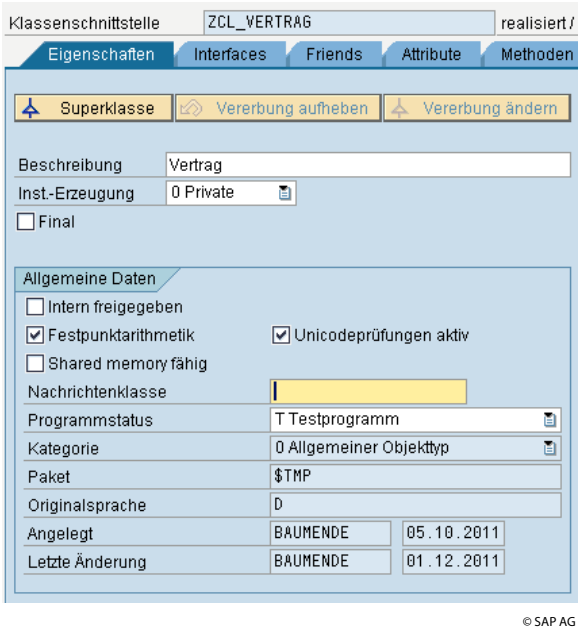


Abb. 5–8
PRIVATE-Instanzerzeugung bei Instanzverwaltung

In vielen Fällen kann dies einfach die Methode FIND_BY_LPOR sein. So wird zunächst sichergestellt, dass es zu einer Datenbankinstanz maximal eine Objektinstanz gibt. Die im Kapitel 11 vorgestellte Methode GENERICINSTANTIATE für Klassenreferenzen im Workflow ruft für die Instanziierung immer die Methode FIND_BY_LPOR auf, die die Instanzverwaltung berücksichtigt.

Nun muss sichergestellt werden, dass diese einzige Objektinstanz immer synchron mit dem Datenbankstand gehalten wird. Das ist dann relativ einfach möglich, wenn die Methoden der Klasse selbst die DB ändern. Dann können sie die Änderung nach COMMIT WORK 1:1 in die gepufferte Hauptspeicherinstanz übernehmen. Beim nächsten Aufruf von FIND_BY_LPOR wird dann der korrekte Datenstand gezogen. Bei komplexen Datenstrukturen kann es manchmal einfacher sein, statt eines Updates in Attributdaten der Instanz einfach die Objektinstanz via IF_WORKFOW~RELEASE( ) aus der Puffertabelle zu löschen. Dann wird sie beim nächsten FIND_BY_LPOR neu von der Datenbank gelesen.

Schwieriger wird es, wenn externe Anwendungen von außerhalb des Workflows schreibend auf die Datenbank zugreifen. Dann kann die Klasse, die die Objektinstanzen verwaltet, davon ggf. gar nichts merken. Wenn dies erlaubt sein soll, dann muss man dem Workflow die Änderungen (z.B. via Eventsteuerung) mitteilen. Dieser kann dann die betreffende Instanz invalidieren und so ein Refresh erzwingen.