

ABAP Objects Live !

Jürgen Heymann
SAP AG



Contents

- Why ABAP Objects?
- ABAP Objects: A Brief Overview
- Persistence and Transaction Control
- GUI Integration
- Remote Objects



Why ABAP Objects?

- **We want the benefits of Objects within R/3**
 - Encapsulation, reduction of complexity, maintainability
 - Focus on interfaces
 - Reuse of low-level components
 - Interoperability across languages (Java, VB, ...) and object systems (DCOM/CORBA)
 - Foundation for new GUI model & support, Controls/Beans
 - Foundation for patterns and frameworks



Why ABAP Objects?

- **Class Library to subsume BOR and BAPIs**
 - BOR = business object repository
 - BAPIs = business APIs (defined in BOR), intended as functional & stateless interface
- **Kernel-level Objects as base for interoperability**



ABAP Objects vs. ABAP/4

- **Complete integration of a fully featured object model into the ABAP programming language**
- **100% upward-compatible extension of ABAP/4**
 - **Bottom up: use objects in existing ABAP programs (reports, module- and function-pools)**
 - **Top down: call forms and functions from within objects**
 - **All ABAP language constructs are available within objects**
 - **Fully integrated into the ABAP Workbench**
- **Class library for global classes (will absorb BOR with BAPIs)**



Contents

- Why ABAP Objects?
- ➔ **ABAP Objects: A Brief overview**
- Persistence and Transaction Control
- GUI Integration
- Remote Objects



Fundamentals: What is an Object ?

- Objects can have Components :
 - ... Attributes (holding its state),
 - ... Methods (defining its behavior)
 - ... Events (publish-subscribe model)
 - ... and constants, static components etc.
- Components are Public, Protected or Private
- Classical Object Model



An Example

```
CLASS CTruck DEFINITION.  
  PUBLIC SECTION.  
    DATA: VehicleId TYPE I READ-ONLY.  
    METHODS: LoadParcel IMPORTING Parcel TYPE REF TO CParcel,  
             UnloadParcel ...  
  PRIVATE SECTION.  
    DATA: ParcelTab TYPE REF TO CParcel OCCURS 0.  
ENDCLASS.  
  
CLASS CTruck IMPLEMENTATION.  
  METHOD LoadParcel.  
    APPEND Parcel TO ParcelTab.  
    "-- do more stuff ...  
  ENDMETHOD.  
ENDCLASS.
```

```
PROGRAM xy.  
  DATA: Parcel TYPE REF TO CParcel,  
        Truck1 TYPE REF TO CTruck,  
        Truck2 TYPE REF TO CTruck.  
  
  ...  
  "-- get input data for parcel from somewhere ...  
  CREATE OBJECT Parcel.  
  CALL METHOD Parcel->SetPars EXPORTING Weight = In_weight.  
  "-- deal with multiple instances  
  CALL METHOD Truck1->UnloadParcel IMPORTING Parcel = Parcel.  
  CALL METHOD Truck2->LoadParcel( Parcel ).
```

Some Important Points

- **Objects are created dynamically**
 - Storage management, garbage collection
- **Access to objects *via object reference only!***
 - Only and explicit means of dependency
 - Sharing always and only via (object) references (similar to field-symbols; all other ABAP types are value-based!)
- **Internal data hidden from users**
 - Private data accessible only by the object's methods
 - private methods



Component Definitions

- **Attributes...**
 - ...store the internal state of an object (data)
 - ...can be references to other objects
 - ...can be read-only
 - ...can be constants
 - **Virtual attributes: 'Attribute' from the outside, inside the object Set- and Get-methods. Dynamic control of Set-/Get-methods.**

```
{DATA|CLASS-DATA} attr TYPE type
[ VALUE val ]
[ READ-ONLY ]
[ VIRTUAL [ SET-METHOD set-method] [GET-METHOD get-method] ].

CONSTANTS const TYPE type VALUE val.
```

Component Definitions

- **Methods...**
 - ...are operations on objects
 - ...are the only way to change the state of an object (other than public attributes)
 - ...can pass back a return value
 - **No method-name overloading!**

```
{METHODS|CLASS-METHODS} method
[ IMPORTING    ...<list of import  parameters> ]
[ EXPORTING   ...<list of export  parameters> ]
[ CHANGING    ...<list of changing parameters> ]
[ RETURNING   result TYPE t ]
[ EXCEPTIONS  ...<list of exceptions> ].
```

Using Attributes and Methods

```
CLASS c1 DEFINITION.
  PUBLIC SECTION.
    DATA: v1 TYPE I,
           o1 TYPE REF TO c1.
    METHODS: m1 IMPORTING a1 TYPE REF TO c1,
             m2 IMPORTING a1 TYPE REF TO c1
              RETURNING result TYPE I.
  PRIVATE SECTION.
    DATA: v2 TYPE I.
ENDCLASS.
```

```
PROGRAM xy.
  DATA o1 TYPE REF TO c1.
  ...
  "--- attribute can occur anywhere a 'normal variable' can occur
  CREATE OBJECT o1.
  x = o1->v1 + sin( o1-> v1 ).
  CALL FUNCTION 'abc' EXPORTING p1 = o1->v1 ... .

  "--- some method calls ...
  CALL METHOD o1->m1 EXPORTING a1 = o1.
  CALL METHOD o1->m1( o1 ).    "-- short form for 1 exporting arg
  ...
  y = obj1->m2( x ).          "-- result can be used in expressions
  ...
```

Component Definitions

- **Events...**

- ...occur at a particular point in time, e.g. 'change in state of an object'
- ...can be raised to inform other interested objects
- ...can pass parameters

```
EVENTS event  
[ EXPORTING ...<list of export parameters> ].
```



Event Handling

- **Events are handled by classes**

- General publish-subscribe model
- Syntax similar to 'Visual Basic' event handling

- **Event handlers...**

- ...are methods for handling events from other objects
- ...are declared with reference to the event to be handled (signature taken from there)
- ...must be 'registered' explicitly



Event Handling Example

Sender

```
*---- proxy class for GUI control
CLASS CButton DEFINITION.
PUBLIC SECTION.
METHODS: SetLabel
          IMPORTING Txt TYPE ... .
EVENTS: Clicked
          EXPORTING DoubleClick TYPE I.
ENDCLASS.
```

```
CLASS CButton IMPLEMENTATION.
...
METHOD AnyMethod.
...
RAISE EVENT Clicked
EXPORTING DoubleClick = 0.
...
ENDMETHOD.
ENDCLASS.
```

Handler

```
CLASS CWindow1 DEFINITION.
PUBLIC SECTION.
"--- handle events by implementing
"--- event handler methods
METHODS:
  OKClicked FOR EVENT Clicked OF CButton
          IMPORTING DoubleClick,
  CanClicked FOR EVENT Clicked OF CButton.
DATA: OKBtn TYPE REF TO CButton.
...
ENDCLASS.
CLASS CWindow1 IMPLEMENTATION.
METHOD Init.
  CREATE OBJECT: OKBtn, CanBtn.
  SET HANDLER: OKClicked FOR OKBtn,
              CanClicked FOR CanBtn.
ENDMETHOD.
METHOD OKClicked.
  IF DoubleClick = 1. ... ENDIF.
ENDMETHOD.
METHOD CancelClicked.
  ... "---- DoubleClick not visible
ENDMETHOD
ENDCLASS.
```

Class (Static -) Components

- **Static attributes...**
 - ...are data on class level, independent of object / instance
 - ...are 'always there' like global variables / functions, global lifetime, with scope tied to class
- **Static methods...**
 - ...can only access class attributes
 - ...can be called like 'global functions', but are tied to class
- **Static Events...**

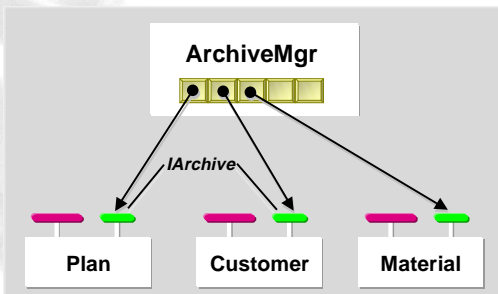
Inheritance

- ABAP Objects has Single Inheritance
 - Add attributes and methods
 - Redefine / override existing methods (in any section) = change implementation (no change in signature)
 - Classes and Methods can be *abstract* and/or *final*

```
CLASS class DEFINITION
  INHERITING FROM superclass.
  ... SECTION.
  "--- added attributes and methods
  DATA: ...
  METHODS: ...
  "--- override / redefine existing method
  METHODS m REDEFINITION.
ENDCLASS.
```

Interfaces

- Interfaces define the interaction between different objects
- Polymorphism independent of class / inheritance
- Classes can implement multiple interfaces
- Uniform access through interface reference



Interface Definition

- Interfaces...
 - ...can define **same components** as class - without implementation
 - ...have separate name spaces for their components
 - ...may 'enclose' multiple other interfaces (hierarchy)
 - Components of enclosed interfaces are not visible in the top-level interface ('black boxes'); there is a mapping / aliasing feature

```
INTERFACE interface.  
  
  [ INTERFACES ...<list of comprised interfaces> . ]  
  
  [ ...<definition of interface components> ]  
  
ENDINTERFACE.
```

Implementation of Interfaces

- A class can implement many interfaces
- Interfaces are implemented 'side-by-side' in a class (like COM)
- No name conflicts on the class level
- No semantic conflicts at class level and interface composition

Using Interfaces

- Access by **interface reference** like object reference
- An interface reference only exposes the components of that interface
- Assignment / 'cast' to another interface possible



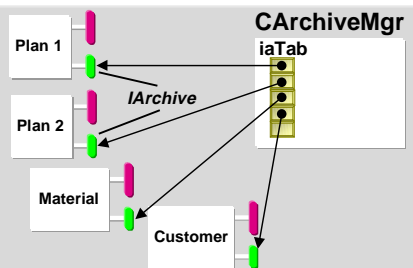
Interface Example

```
INTERFACE IArchive DEFINITION.  
  DATA: ObjID TYPE T_OID READ-ONLY.  
  EVENTS: Saved, ... .  
  METHODS: SaveYourself IMPORTING ... .  
ENDINTERFACE.
```

```
CLASS Customer DEFINITION.  
  INTERFACES: IArchive, IWorkflow, ... .  
ENDCLASS.
```

```
CLASS Customer IMPLEMENTATION.  
  ...  
  METHOD IArchive-SaveYourself.  
    "---- save all own data into ...  
    ...  
    RAISE EVENT IArchive-Saved ... .  
  ENDMETHOD.  
  ...  
ENDCLASS.
```

```
CLASS CArchiveMgr DEFINITION.  
  PUBLIC SECTION.  
  DATA: IAObj TYPE REF TO IArchive.  
  DATA: IATab TYPE REF TO IArchive  
         OCCURS 0.  
  ...  
  METHOD DoArchive.  
    "---- archive all objects in table  
    LOOP AT IATab INTO IAObj.  
      WRITE: / "Wrote:", IAObj->ObjID.  
      CALL METHOD IAObj->SaveYourself ... .  
    ENLOOP.  
  ENDMETHOD.
```



Nested Interfaces

- **Interfaces can be nested (composition)**
 - Components of enclosed interfaces are not visible in the top-level interface ('black boxes')
 - No name conflicts? (we have all components in interfaces and no overloading)
 - No semantic conflicts
- **ALIASES**
 - Selectively make components visible on the higher levels
 - Allows new clean abstractions
 - No automatic ALIASES (no 'deferred name conflicts')



Contents

- Why ABAP Objects?
- ABAP Objects: A Brief Overview
- ➔ **Persistence and Transaction Control**
- GUI Integration
- Remote Objects



Persistence Model: Overview

- **Requirements**

- Support existing tables and RDBs well now, later OODBs
- Mapping of objects to relational DBs (later OODBs?), mapping foreign keys <=> references
- Efficient loading and updating from/to the DB
- Integrated Transaction Control, Locking, 'consistency' checks
- Flexible implementation: Cover >80%, leave room for special cases

- ⇒ **Persistence / Transaction Framework**

- Phase 1: Generated code (class methods) for persistent classes



Persistent Objects

- **Logical (Business) Objects exist unique on Database**
 - Identity tied to key(s)
- **We want unique logical objects also in memory**
- **'Persistent Object'**
 - in-memory incarnations of 'objects' on DB
 - loaded under control of persistence manager
 - at commit, all changes to persistent objects are also made persistent on the DB
 - there can be transient instances of persistent objects



Mapping Objects to DB Tables

- Specify mapping of attributes to DB tables and fields
 - A class can map to multiple DB tables
- Navigation, i.e. map foreign keys to references
 - issues: partial keys & context
 - Navigation methods (later possibly virtual proxies)
- Read through View
- Updates to DB table directly
 - array update at end of transaction
- Loading explicit (OQL-like) or on-demand by navigation



What could we want to write?

```
*--- definition of persistent class
CLASS CCust DEFINITION PERSISTENT.
PUBLIC SECTION.
    DATA: k1 TYPE ... KEY-FIELD,    "-- key-field implies read-only
           a1 ...                    "-- normal attributes

*--- this information kept external:
* DB-Mapping: k1 to dbtab1-f2, a1 to dbtab2-f4, ...
* Associated view:
...
ENDCLASS.

*--- load objects into memory (instance manager)
SELECT OBJECTS CCust WHERE k1 = 'A*'.
    "-- short for: CALL METHOD CCust=>LoadInsts ( k1 = 'A*' ).

*--- load and fill result set as well
SELECT OBJECTS CCust WHERE k1 = 'A*' INTO CustSet.

DATA o1 TYPE REF TO CBukrs.
*-- map key to ref, load if needed
o1 = CCust=>GetInst( 'ABC*' ).

o1 = 'ABC*'.                "-- associated key is compatible with reference?
```



Which objects need update?

- **Explicit 'change' Mode**
 - No 'watching all attribute changes ...'
 - Objects are 'read-only' initially (conceptually)
 - Objects call 'SetInWork()' to go into 'change' mode
 - SetInWork() puts an object into the WorkSet of this transaction (then 'current' state is saved)
- **At Commit, the TAMgr goes through all objects in WorkSet and writes the changes back to the DB (array updates)**
- **In-Memory transactions: 1-level rollback for objects (restore saved state) and checkpoints (update saved state)**



Transaction Control

- **'Commit' is explicit**
 - possible nesting protocol for process objects
 - 'Commit' triggers active check for consistency
- **Checking 'consistency'**
 - no 'Complete' tag on each object
 - Process objects vs. data objects
 - 'Check hierarchy' is independent
- **After all checks: Persistence Manager writes back all changes to persistent objects**
 - INSERT is a special case (special for DB)
 - DELETE of persistent object is explicit



Other Issues

- Update tasks (V1 and V2)
- Locking
 - hierarchical locking service
 - locking service buffer for scalability



Contents

- Why ABAP Objects?
- ABAP Objects: A Brief Overview
- Persistence and Transaction Control
- GUI Integration
- Remote Objects



GUI Integration

- **Until now...**
 - Screens (Dynpros) were tied to global data
 - DDIC functionality tied to GUI only
 - ◆ certain checks
 - ◆ help text, labels
 - Modules (program sections) to write interaction code
 - ◆ 'Module Pool' special program type
 - ◆ 'Modules' as entry points
 - ◆ Controlled by DYNP Controller under dialog events (PAI, PBO,...)



GUI Integration

- **What do we want?**
 - Display / Edit objects in the (SAP) GUI
 - Goal: 'Objects display / edit themselves'
 - Local and global classes can be used
 - Independent of channel (GUI, ALE, batch-input, ...)
 - All checks within the classes
 - Reuse & combine 'OO Screens'
 - Don't lose anything we have already...
 - ◆ ASAP = as simple as possible



OO Screens

- **'OO Screen' = (Sub-) Screen tied to object**
 - Field references 'obj->attr'; obj = 'current object'
 - Methods handle GUI events (instead of modules)
 - Screens also define which methods get called
- **Multiple screens can be defined for classes and interfaces**
- **Screens can be nested arbitrarily**
- **'read-only' vs. 'edit screens': just show values, one big check, or fine grained interaction code**

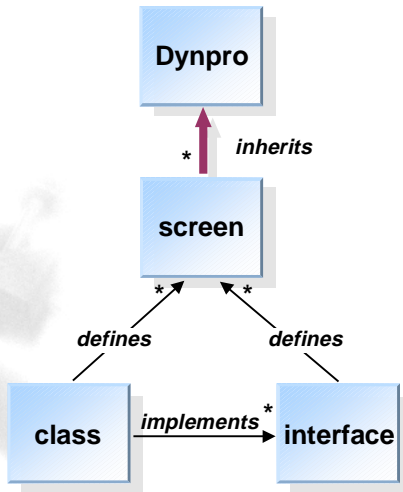


Screens become Objects

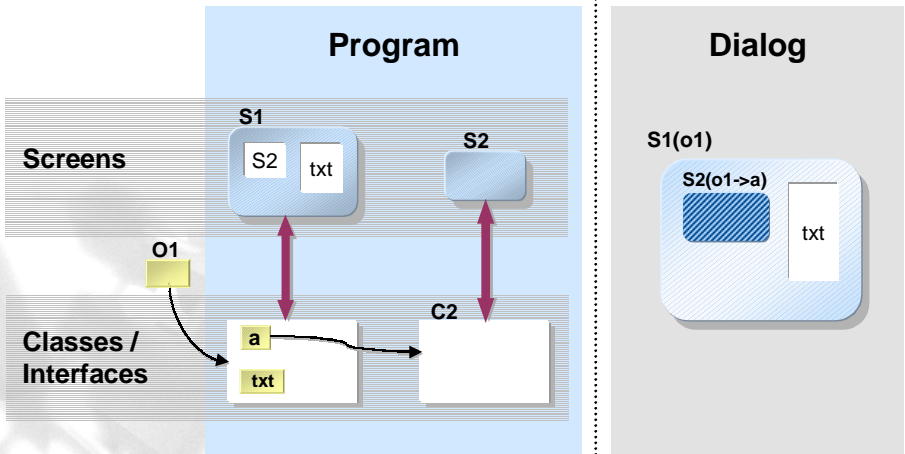
- **Screens become Objects themselves**
- **Class 'Dynpro' encapsulates GUI control functionality**
- **All Screens inherit from Dynpro; can redefine certain methods (display, ...)**
- **Formerly global functionality and data incorporated into Dynpro**
 - **SCREEN, CALL/SET/LEAVE SCREEN,**



OO GUI: Static View



OO GUI: Dynamic View



Contents

- Why ABAP Objects?
- ABAP Objects: A Brief Overview
- Persistence and Transaction Control
- GUI Integration
- Remote Objects



Remote Objects

- What we need
 - Create objects on remote destination
 - Call methods on remote objects / interfaces
 - Static typing also for remote objects
 - Versioning
 - *Possibly remote calls visible in code!*
- Approach
 - Generated proxies and stubs



Remote Objects

- **Interfaces (&Classes?) can be defined 'REMOTABLE'**
 - **Public: Methods only (possibly events), no attributes**
 - **RFC restrictions on argument types**
 - **Get GUIDs etc., versioning**
 - **force 'REMOTE' tag on method call**
- **In general: Only interfaces for remote access**
 - **Hides proxy class (local and remote look the same)**
 - **Target can select implementing class**



Remote Objects: Use

- **CREATING ...**
 - **CREATE OBJECT ... TYPE (name) DESTINATION ...**
- **INTERFACE ... DEFINITION REMOTABLE.**
- **'Remote' Method call**
 - **CALL REMOTE METHOD remote_iref->m**
 - **'possibly remote' explicit**
 - **also includes DB-Commit (even in local case)**
- **IF ref IS REMOTE ...**



Summary

- ABAP Objects Language now pretty complete
- Activity Areas:
 - Persistence, Transactions, Update task
 - GUI Integration
 - Remote Objects
- Basic 'Object Application Framework'
- Test in pilot projects, then refine...



Discussion

Questions (!)

&

Answers (?)

