

ARCH-9: Using Object-oriented ABL Features in N-Tier Environments

Mike Fechner, Consultingwerk Ltd.

Agenda

- Object orientation in the ABL
- Usage of classes on the AppServer™
- Using proxy procedures
- Keeping objects persistent on the AppServer
- Object serialization
- Exchanging objects with non OpenEdge® applications

Object orientation in the ABL

- Available since 10.1A, enhanced in 10.1B
- 100% compatible with procedural programming
 - Procedures may instantiate classes and execute methods
 - Classes may run procedures
 - Classes may be passed as parameters to procedures
- Available on all ABL runtimes (C/S, AppServer)

Object orientation in the ABL

- Enhance ABL with new Programming model including
 - Strong typing
 - Compile time checks
 - Inheritance / Overloading / Polymorphism
 - Interfaces
- Common coding concepts available in modern development languages

Strong typing

- Classes enhance the type model of ABL as data types
- A class can be used like a data type
- Compile time check for validity of properties and methods and their parameters
- The chance that code accessing classes actually runs as required when it compiles is much higher than with procedural code due to compile time validation (less runtime errors)

Inheritance / Overloading / Polymorphism

- Classes may inherit from other classes offering a richer, more specialized variant
 - *Invoice* may inherit from the *SalesDocument* class offering additional functionality
- Method overloading allows reuse of a method name in more specialized class definition
- Polymorphism allows the creation of methods with the same name but different signatures
 - i.e. the class itself handles different data types or defaults passed, not the caller

Demo

- Introducing the *SalesDocument* class and the *Invoice* class used in the samples later in this presentation

Interfaces

- An interface defines minimum requirements for a class, specified at class definition
- Describes the methods (including signature) a class (or SUPER-classes) must implement
- A parameter to a method or procedure may be of an Interface type allowing each class implementing the interface to be passed
- May serve as contract between developer of calling class/procedure and developer of implementing class

Interfaces

- Classes implementing an interface do not require a same base class to be passed as a parameter of the interface type
- When a called procedure/method expects an interface type, this procedure/method only knows the methods of the interface type, the other methods are shielded
- Sample later in this presentation: ISerializable

Sounds nice, but what's missing?

- Out of the box usage over the AppServer boundary
- Reflection
 - Allowing to query a classes methods and attributes at runtime (as available for procedures)
 - Dynamic method calls, dynamic instantiation
 - Would make the implementation of Serialization much easier (later in this talk)

Agenda

- Object orientation in ABL
- Usage of classes on the AppServer
- Using proxy procedures
- Keeping objects persistent on the AppServer
- Object serialization
- Exchanging objects with non OpenEdge applications

Usage of classes on the AppServer

- Classes can be used on the AppServer and on any other ABL runtime environment
- The AppServer protocol/server handle does not support classes
 - No remote invocation of classes over the AppServer boundary
 - Can't pass a class as an parameter to a remote procedure

No support for remote invocation

- Client is unable to create instance on AppServer
- At first sight looks like a real show stopper
- But it shouldn't be high on an architects wish list anyway
 - Classes behave similar to persistent procedures at run time
 - Would require state-aware or stateless-bound
 - One call for constructor, second call for method call, third call for destructor
 - Wouldn't scale well in most environments

Classes as parameters to the AppServer

- Within a session classes are passed as object references (strong-typed handles) to methods and procedures passed
- Instances are only valid within a single session (like Widget handles)
- Other OO environments support marshalling or serialization of objects when passed from one runtime to another

Classes as parameters to the AppServer

```

SalesDocument.cls RemoteParameter.p
/* ***** Definitions ***** */
USING samples.*.

DEFINE VARIABLE mySalesDocument AS SalesDocument NO-UNDO.
DEFINE VARIABLE hServer AS HANDLE NO-UNDO.

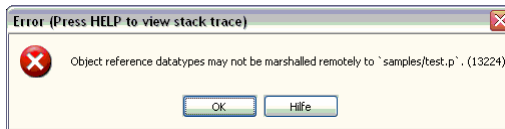
/* ***** Main Block ***** */

mySalesDocument = NEW SalesDocument () .
CREATE SERVER hServer.
hServer:CONNECT ("AppService Exchange2007") .
RUN samples/test.p ON hServer (mySalesDocument)

```

Class instantiated on client

Attempt to pass to server procedure



Classes as parameters to the AppServer

- ABL does not support marshalling of classes over the AppServer
- It might be problematic if a class requires DB connections, buffer scope, transaction scope
- IMHO more critical restriction than lack of remote invocation
- Possible solution: Serialization of classes later in this talk

Agenda

- Object orientation in ABL
- Usage of classes on the AppServer
- Using proxy procedures
- Keeping objects persistent on the AppServer
- Object serialization
- Exchanging objects with non OpenEdge applications

Usage of proxy procedures

- Procedures are the only entry point to the logic running on the AppServer
- As persistent procedures limit scalability of the AppServer, non persistent .p files should be used to access any functionality on the AppServer
 - Procedural
 - Object oriented
- Signature restricted to ABL data types, temp-tables, ProDatasets as parameters, ...

Consistent across sample implementations

- Simple .p files as proxies to persistent procedures on the AppServer:
 - Procedural reference architecture implementation (service interface)
 - ADM2 SmartDataObject (SDO) access
 - Dynamics manager access
- Simple .p files as proxies to classes:
 - Class based reference architecture implementation

Proxy procedures

- Required for each method exposed to the client
- Enabling consistent error handling for all calls to the AppServer (i.e. ERROR-STATUS, RETURN-VALUE)
- Use SESSION:EXPORT() to ensure that only these procedures are callable by clients (i.e. procedures in a designated proxy directory, based on client authentication and status)
- Defines maximum transaction scope

Demo

- InvoiceTotal method of Invoice class (samples/callInvoiceTotal.p)
- Generic Interface to Invoice class (samples/genericInvoiceInterface.p)
- Proxy class to be used on client (samples/proxyInvoiceClass.p)

Clean up your garbage!

- Progress does not remove the object instance from memory! Expect errors or stop conditions:

```
DO ON ERROR UNDO, LEAVE
  ON STOP UNDO, LEAVE:
    myInvoice = NEW Invoice(piInvoiceNum).

    /* Code that uses the object follows */
END.

IF VALID-OBJECT(myInvoice) THEN
  DELETE OBJECT myInvoice.
```

Proxy Procedures: Summary

- Proxy Procedures enable access to class based logic from a client
- Proxy Procedures need to know all methods exposed to clients
- May control authentication, error-handling, transaction scope

Agenda

- Object orientation in ABL
- Usage of classes on the AppServer
- Using proxy procedures
- Keeping objects persistent on the AppServer
- Object serialization
- Exchanging objects with non OpenEdge applications

Persistent objects / class instances

- A class instance running on the AppServer remains in memory until it is deleted using the DELETE OBJECT statement (similar to persistent procedures)
- A pre-initialized class can be re-used within subsequent client requests
- With state-less or state-free AppServer, each call may be run by a different client
- Data inside class not unique to a client / user

Allocation of running class instances

- Using SESSION:FIRST-OBJECT, NEXT-SIBLING attributes (as used in class based reference architecture implementation)
- Using Manager Temp-Tables with Progress.Lang.Object fields (my personal preference), shall be encapsulated inside procedure or class, requires cast
- Using GLOBAL SHARED variables of type Progress.Lang.Object or a specialized class or interface type (can't be used from within objects)

Sample SESSION:FIRST-OBJECT

- **Placeholder for sample:**
 - allocating a class instance
 - casting to required class

Sample Manager Temp-Table

- **Placeholder for sample:**
 - Definition of manager temp-table
 - Class Name, Instanceld,
InstanceReference
 - Allocating Class by ClassName or
ClassName and Instanceld
 - CAST InstanceReference

Agenda

- Object orientation in ABL
- Usage of classes on the AppServer
- Using proxy procedures
- Keeping objects persistent on the AppServer
- **Object serialization**
- Exchanging objects with non OpenEdge applications

Data members (ABL)

- Classes consist of an implementation (source code, R-code) and data members
- Implementation of a class is sealed at runtime
- Data members describe the current state of a class instance
 - “Simple” data types (CHAR, INT, DATE, ...)
 - Class data types
 - Temp-Tables, ProDatasets
 - Database Record Buffers

Object Serialization (en.wikipedia.org)

- In computer science, in the context of data storage and transmission, **serialization** is the process of saving an object onto a storage medium (such as a file, or a memory buffer) or to transmit it across a network connection link, either in binary form, or in some human-readable text format such as XML. The series of bytes or the format can be used to re-create an object that is identical in its internal state to the original object (actually a clone).
- This process of serializing an object is also called **deflating** or **marshalling** an object. The opposite operation, extracting a data structure from a series of bytes, is **deserialization** (which is also called **inflating** or **unmarshalling**).

Serialization/Deserialization

- Serialization describes the process of building a representation of a class instance describing the **current state** of an object
- Deserialization describes the process of using that state information and building a **new instance** of an object with the same state
- Multiple formats available
 - Binary for speed, XML for portability
 - Custom formats (Temp-Tables, ProDatasets)

Serialization/Deserialization

- ABL does not implement any Serialization at all
- XML common serialization format
 - .NET (XMLSerialization class, using Reflection)
 - Document style Web services, complex types in XML schema

Serialization use cases

- Passing classes as parameters to/from AppServer procedures (XML to LONGCHAR)
- Business objects wrapping ProDatasets as well as additional data members
- Storing classes in a context database table (i.e. complex session information) or AppServer CONNECTION-CONTEXT
- Passing classes as parameters to Web services

Serialization samples

- Sample of serializing the *SessionInformation* class
- Sample of serializing the *Invoice* class

Classes supporting Serialization

- Consultingwerk.Lang.IXMLSerializable
 - Interface containing `Serialize()` method
- Consultingwerk.Lang.XMLSerializable
 - Helper methods
- Consultingwerk.Lang.XMLDocument, XMLNodeReference
 - Typed references to X-DOCUMENT, X-NODEREF handles, support distinction on method signatures (not given by HANDLE)

Supporting methods

- VOID **Serialize** (XMLNodeReference)
 - Code this as required by the data members of your class
 - Use InsertPropertyNode (next slide) to store values for any data member
 - Include SUPER:Serialize(XMLNodeRef.)
- XMLDocument **Serialize** ()
 - Implemented in XMLSerializable
 - Starts Serialization to a new XML Document

Supporting methods

- **InsertPropertyNode** (XMLNodeReference, PropertyName, <anytype> PropertyValue)
 - Adds a node for a single data member (PropertyName) to the XML document
 - XMLNodeReference contains X-NodeRef to object node in X-Document
 - Multiple implementations in XMLSerializable to support “any” possible data type (including IXMLSerializable for classes)

Cascading Serialization

- A SUPER class (the class, the *Invoice* class inherits from) might contain data members that require serialization
 - Give the SUPER class a chance to `Serialize()` data members
- Data members might be of class types, so they might require `Serialization` as well (including SUPER classes)

Sample for cascading Serialization

- *Invoice* class inherits from *SalesDocument*
 - *SalesDocument* has the following members:
 - *DocumentNumber* AS INTEGER
 - *DocumentAddress* AS *AddressInformation*
 - *Invoice* has the following member:
 - Buffer *bInvoice* (Table sports2000.invoice)
 - Buffer will be serialized using helper class, storing TABLE, ROWID and a SHA1 hash of the current value

Sample for cascading Serialization

```

<?xml version="1.0" ?>
- <Invoice>
  <bInvoice DBNAME="sports2000"                                BUFFER bInvoice
    ROWID="0x000000000000006f"
    SHA1="NVWqfsIOTW8IXfAoU9ZhgOwJIm8="
    TABLE="Invoice" />
  <DocumentNumber>2</DocumentNumber>                        SUPER SalesDocument
  <DocumentAddress>
    <Name>Off The Wall</Name>
    <Contact>Nicole Beller</Contact>
    <Address>20 Leedsville Ave</Address>
    <Address2 />
    <City>Export</City>
    <State>PA</State>
    <PostalCode>15632</PostalCode>
    <Country>USA</Country>
  </DocumentAddress>
</Invoice>

```

Deserialization

- Process of populating the data members of a class with the information from the XML document
- Either during the constructor of the class or using a method call
 - Constructor will always create a new instance
 - Multiple constructors allowed
 - Custom method call might reuse a persistent instance from another AppServer request

Deserialization, cont.

- Keep cascaded documents in mind
- Deserialize class type members as well as SUPER class members
- Walk though the direct child nodes of the given node and populate data members (see CASE statement in sample)
- When nodes represent class type data members, use their Deserialization routine

Deserialization, sample

```

DO i = 1 TO hObjectNode:NUM-CHILDREN :
  hObjectNode:GET-CHILD (hNode, i) .

  IF hObjectNode:NUM-CHILDREN <> 1 THEN NEXT.

  hObjectNode:GET-CHILD(hObjectNode, 1) .

  CASE hObjectNode:NAME:
    WHEN „DocumentNumber“ THEN
      ASSIGN iDocumentNumber
              = hObjectNode:NODE-VALUE .
    WHEN „DocumentAddress“ THEN
      ASSIGN documentAddress = NEW
              DocumentAddress(hObjectNode) .

  END CASE.
END .

```

Keep basic rules of ABL in mind!

- A client not connected to a database can't create a class instance that requires DB tables
 - RCODE-INFO:DB-REFERENCES
- Transactions and Record-Locks can't be serialized
- When re-fetching a row, the row may be locked or changed by another user

Serialization/Deserialization samples

- Storing the SessionInformation in AppServer session context
- Passing InvoiceEntity class between client and AppServer between method calls

Agenda

- Object orientation in ABL
- Usage of classes on the AppServer
- Using proxy procedures
- Keeping objects persistent on the AppServer
- Object serialization
- Exchanging objects with non OpenEdge applications

XML Serialization with .NET Proxies

- Sample using XMLSerializer class in C#
- Create C# class that matches the InvoiceEntity object

XML Serialization with Web service calls

- bprowsldoc lists XSD complex types as LONGCHAR parameters (INPUT or OUTPUT)
- XML serialized objects can be used here
- Build objects that match the complex types

Summary

- Object orientation in ABL is available today
- OO in ABL offers greater control over your code (strong typing, inheritance, interfaces, private/public access)
- Usage of classes on the AppServer is possible
- Serialization is possible
- Even with some features missing (reflection, native Serialization) there's no reason to hesitate
- Design your classes with possible enhancements to OOABL in mind

Questions

- Don't hesitate to ask...
- Now or send email later:
mike.fechner@consultingwerk.de



For more information

- Exchange 2007 Sessions:
 - DEV-6: Getting started with Object-oriented Programming
 - DEV-12: Object-oriented Programming in the OpenEdge ABL
 - ARCH-7: A Class-based Implementation of the OERA
- 10.1B Object-oriented Programming manual
- 10.1B Working with XML manual
- Sample code download on PSDN,
<http://www.consultingwerk.de/exchange2007/>

Thank you!

- I appreciate your time and interest