



Objektorientiert auf dem AppServer


Objektorientierung in der ABL, 24.08.2007
Mike Fechner, Consultingwerk Ltd.
<http://www.consultingwerk.de>



Mike Fechner, Consultingwerk Ltd.

- Unabhängige EDV Beratung
- Sitz in der Kölner Altstadt
- Kunden in Deutschland, Europa, USA
- 17 Jahren Progress Erfahrung


Objektorientiert auf dem AppServer 2



Agenda

- Objektorientierung in der ABL
- Verwendung von Klassen auf dem AppServer™
- Proxy Prozeduren
- Objektpersistenz auf dem AppServer
- Serialisierung
- Objektaustausch mit nicht OpenEdge® Anwendungen


Objektorientiert auf dem AppServer 3



Objektorientierung in der ABL

- Verfügbar seit 10.1A, erweitert in 10.1B
- 100% kompatibel mit prozeduraler Programmierung
 - Procedures können Klassen instanziiieren und Methoden ausführen
 - Klassen können Prozeduren ausführen
 - Instanzen können als Parameter an Prozeduren übergeben
- Verfügbar auf allen ABL Clients (C/S, WebClient AppServer, WebSpeed)


Objektorientiert auf dem AppServer 4



Objektorientierung in der ABL

- Erweitert die ABL mit zusätzlichen Programmiermodellen, incl.
 - Strong typing
 - Compilezeit Prüfungen
 - Vererbung / Überladung / Polymorphismus
 - Interfaces (Schnittstellen)
- Verbreitete Programmierkonzepte, verfügbar in den gängigen modernen Entwicklungsumgebungen


Objektorientiert auf dem AppServer 5



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)


Objektorientiert auf dem AppServer 6



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


Objektorientiert auf dem AppServer 7



Demo

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


Objektorientiert auf dem AppServer 8



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

Objektorientiert auf dem AppServer 9



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

Objektorientiert auf dem AppServer 10

Consultingwerk
software architecture and development

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)

Objektorientiert auf dem AppServer 11

Consultingwerk
software architecture and development

Agenda

- Objektorientierung in der ABL
- **Verwendung von Klassen auf dem AppServer**
- Proxy Prozeduren
- Objektpersistenz auf dem AppServer
- Serialisierung
- Objektaustausch mit nicht OpenEdge Anwendungen


Objektorientiert auf dem AppServer 12



Verwendung von Klassen auf dem AppServer

- Klassen können aus dem AppServer wie in jeder ABL Laufzeit Umgebung genutzt werden
- Das AppServer Protokoll/Server Handle unterstützen aber keine Klassen
 - Kein entfernter Aufruf über die AppServer Boundary möglich
 - Eine Klasse kann nicht als Parameter an eine entfernte Prozedur übergeben werden

Objektorientiert auf dem AppServer 13



Kein entfernter Aufruf

- Client kann keine Instanz auf dem AppServer erzeugen
- Auf ersten Blick eine kritischen Einschränkung
- Sollte aber nicht weit oben auf der Wunschliste eines Anwendungsarchitekten stehen
 - Laufzeitverhalten von Klassen ähnlich wie das persistenter Prozeduren
 - State-aware oder Stateless-bound wäre erfordert
 - Ein Call für Konstruktor, 2. Call für Methodenaufruf, 3. Call für Destruktor
 - Skalierbarkeit daher fragwürdig

Objektorientiert auf dem AppServer 14

Consultingwerk
software architecture and development

Klassen als Parameters an den AppServer

- Innerhalb eine Session werden Objekte als Objekt Referenzen (typisierte Handles) an Methoden und Prozeduren übergeben
- Instanzen haben aber nur innerhalb einer Sitzung Gültigkeit (wie Widget Handles)
- Andere OO Umgebungen unterstützen das Marshalling oder Serialisierung von Objekten um diese zwischen verschiedenen Umgebungen zu übergeben

Objektorientiert auf dem AppServer 15

Consultingwerk
software architecture and development

Klassen als Parameters an den AppServer


```
/* ***** 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)
```

Instanz auf dem Client erstellt

Übergabe an Serverprozedur

Error (Press HELP to view stack trace)
Object reference datatypes may not be marshalled remotely to 'samples/test.p', (13224)


Objektorientiert auf dem AppServer 16



Klassen als Parameters an den AppServer

- ABL unterstützt klein “marshallen” von Klassen über das AppServer Protokoll
- Problematisch, falls Klassen DB Verbindungen, Buffer/Locking/Transaction Scopes verwendet
- Deutlich kritischere Einschränkung als das Fehlen der entfernten Aufrufe
- Mögliche Lösung: Serialisierung (am Nachmittag)


Objektorientiert auf dem AppServer 17



Agenda

- Objektorientierung in der ABL
- Verwendung von Klassen auf dem AppServer
- **Proxy Prozeduren**
- Objektpersistenz auf dem AppServer
- Serialisierung
- Objektaustausch mit nicht OpenEdge Anwendungen


Objektorientiert auf dem AppServer 18



Proxy Prozeduren

- Prozeduren sind der einzige Einstiegspunkt zu Logik auf dem AppServer
- Da persistente Prozeduren ebenfalls kritisch für die Skalierbarkeit einer Anwendung sind (stateless-bound), sollten nicht persistente .p Files als Schnittstelle zu sämtlicher Logik auf dem AppServer verwendet werden
 - prozedural
 - objektorientiert
- Signatur darf nur ABL Datentypen, Temp-Tables, ProDatasets, ... als Parameters enthalten

Objektorientiert auf dem AppServer 19



Beispiel Implementierungen

- Einfache .p files als Proxies zu persistenten Prozeduren auf dem AppServer:
 - Procedurale OERA (Service Interface)
 - ADM2 SmartDataObject (SDO) API
 - Dynamics Manager Zugriff
- Einfache .p Files als Proxies zu Klassen:
 - Class based reference architecture implementation (OO-OERA, O3RA)

Objektorientiert auf dem AppServer 20

Consultingwerk
software architecture and development

Proxy Prozeduren

- Notwendig für jede Methode, die dem Client zur Verfügung gestellt werden soll – keine Dynamik
- Ermöglichen konsistente Fehlerbehandlung (z.B. ERROR-STATUS, RETURN-VALUE)
- Über SESSION:EXPORT() kann sich gestellt werden, dass nur diese Prozeduren aufrufbar sind (z.B. Prozeduren in einem designiertem Proxy Verzeichnis, abhängig von Client Authentifizierung und Status)
- Bestimmen den maximalen Transaktionsradius

Objektorientiert auf dem AppServer 21

Consultingwerk
software architecture and development

Demo

- InvoiceTotal Methode der Invoice Klasse (samples/callInvoiceTotal.p)
- Generischer Proxy für die Invoice Klasse (samples/genericInvoiceInterface.p)
- Proxy Klasse für den Client (samples/proxyInvoiceClass.p)
- Interface Implementierung durch die Proxy Klasse

Objektorientiert auf dem AppServer 22

Consultingwerk
software architecture and development

Räumen Sie hinter sich wieder auf!

- Progress entfernt Objekt Instanzen nicht selbständig aus dem Speicher! Erwarten Sie Error- oder Stop Konditionen:

```
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.
```


Objektorientiert auf dem AppServer 23

Consultingwerk
software architecture and development

Proxy Prozeduren: Zusammenfassung

- Proxy Prozeduren ermöglichen Zugriff auf objektorientierte Logik von einem Client
- Proxy Prozeduren müssen sämtliche dem Client zur Verfügung gestellten Methoden kennen (strong typing, keine Dynamik)
- Ermöglichen Kontrolle von Authentifizierung, Fehler-Behandlung, Transaktionsradius


Objektorientiert auf dem AppServer 24



Agenda

- Objektorientierung in der ABL
- Verwendung von Klassen auf dem AppServer
- Proxy Prozeduren
- **Objektpersistenz auf dem AppServer**
- Serialisierung
- Objektaustausch mit nicht OpenEdge Anwendungen

Objektorientiert auf dem AppServer 25



Persistente Objekte / Instanzen

- Eine Objektinstanz auf dem AppServer bleibt so lange im Speicher, bis sie explizit über das DELETE OBJECT Statement gelöscht wird – vergl. persistente Prozeduren
- Einmal initialisierte Klasse kann von nachfolgenden Client Requests wieder verwendet werden
- Bei state-less oder state-free wahrscheinlich im Kontext eines anderen Users oder Clients
- Daten Mitglieder in einer Klasse sind somit nicht einem Client / User zuzuordnen

Objektorientiert auf dem AppServer 26

Consultingwerk
software architecture and development

Auffinden laufender Objektinstanzen

- Über SESSION:FIRST-OBJECT, NEXT-SIBLING Attribute (so z.B. in der O3RA=OO-OERA), **CAST erforderlich**
- Über Manager Temp-Tables mit Progress.Lang.Object Feldern (Meine persönliche Präferenz), sollte über eine Prozedur oder Klasse gekapselt werden, **CAST erforderlich**
- Über GLOBAL SHARED Variablen vom Datentyp Progress.Lang.Object oder einer speziellen Klasse oder Interface, **kann nicht aus Klassen deklariert werden**


Objektorientiert auf dem AppServer 27

Consultingwerk
software architecture and development

Beispiel SESSION:FIRST-OBJECT

- Auffinden einer Objektinstanz
 - GetClass():TypeName
 - TYPE-OF Funktion
- CAST auf die eigentliche Klasse


Objektorientiert auf dem AppServer 28

 Consultingwerk
software architecture and development

Beispiel GLOBAL SHARED VARIABLE

- Zugriff auf Objekte über GLOBAL SHARED VARIABLEN
- Vorteil: Kein CAST erforderlich
- Nachteil: Nur in Prozeduren nutzbar
- Einfachste Implementierung für Singletons (Klassen mit nur einer Instanz)


Objektorientiert auf dem AppServer 29

 Consultingwerk
software architecture and development

Beispiel Manager Temp-Table

- Definition der Manager Temp-Table
 - Name der Klasse, Instanceld, InstanceReference
- Auffinden der Instanz über Namen der Klasse oder Instanceld
- CAST InstanceReference


Objektorientiert auf dem AppServer 30



Agenda

- Objektorientierung in der ABL
- Verwendung von Klassen auf dem AppServer
- Proxy Prozeduren
- Objektpersistenz auf dem AppServer
- **Serialisierung**
- Objektaustausch mit nicht OpenEdge Anwendungen


Objektorientiert auf dem AppServer 31



Daten Mitglieder (ABL)

- Klassen bestehen aus einer Implementierung (Quellcode, R-code) und Daten Mitgliedern
- Implementierung zur Laufzeit eingefroren
- Daten Mitglieder beschreiben den gegenwärtigen Zustand einer Instanz
 - “Einfache” data types (CHAR, INT, DATE, ...)
 - Klassen Daten Typen
 - Temp-Tables, ProDatasets
 - Datenbank Record Buffers


Objektorientiert auf dem AppServer 32



Serialisierung (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**).


Objektorientiert auf dem AppServer 33



Serialisierung/Deserialisierung

- Serialisierung beschreibt den Prozess eine Darstellung einer Instanz zu bilden, die ihren **aktuellen Status** beschreibt
- Deserialisierung beschreibt den Prozess diese Zustandsbeschreibung zu verwenden und eine **neue Instanz** mit identischem Zustand zu erzeugen
- Mehrere Formate üblich
 - Binär für Performance, XML für Portabilität
 - Sonst. Formate (z.B. Temp-Tables, ProDatasets)


Objektorientiert auf dem AppServer 34



Serialisierung/Deserialisierung

- ABL implementiert (derzeit) gar keine Serialisierung
- XML gängiges Format für Serialisierung
 - .NET (XMLSerialization Klasse mit Reflection, dynamisch)
 - Document style Web services, Komplexe Typen in XML schema

Objektorientiert auf dem AppServer 35



Anwendung von Serialisierung

- Übergabe von Klassen als Parameters an/von AppServer Prozeduren (XML zu LONGCHAR)
- Business Objekte die ProDatasets sowie zusätzliche Daten Mitglieder Kapseln
- Ablage von Klassen in einer Context Datenbank (z.B. Session Information) oder dem AppServer CONNECTION-CONTEXT
- Übergabe von Klassen als Parameters an Web Services

Objektorientiert auf dem AppServer 36

Consultingwerk
software architecture and development

Serialisierung, Beispiele

- Beispiel zum Serialisieren der *SessionInformation* Klasse
- Beispiel zum Serialisieren der *Invoice* Klasse


Objektorientiert auf dem AppServer 37

Consultingwerk
software architecture and development

Hilfsmethoden

- Consultingwerk.Lang.IXMLSerializable
 - Interface mit Serialize() Methode
- Consultingwerk.Lang.XMLSerializable
 - Hilfsmethoden
- Consultingwerk.Lang.XMLDocument, XMLNodeReference
 - Typisierte Referenzen auf X-DOCUMENT, X-NODEREF Handle, zur Unterscheidung in Methoden Signaturen (nicht gegeben bei HANDLE)


Objektorientiert auf dem AppServer 38

 Consultingwerk
software architecture and development

Hilfsmethoden

- **VOID Serialize** (XMLNodeReference)
 - Nach Bedarf zu implementieren (für die Daten Mitglieder einer Klasse)
 - InsertPropertyNode (nächste Folie) um die Werte eines Daten Mitgliedes abzulegen
 - SUPER:Serialize(XMLNodeRef)
- **XMLDocument Serialize** ()
 - Implementiert in XMLSerializable
 - Eröffnet die Serialisierung in ein neues XML Document

Objektorientiert auf dem AppServer 39

 Consultingwerk
software architecture and development

Hilfsmethoden

- **InsertPropertyNode** (XMLNodeReference, PropertyName, <anytype> PropertyValue)
 - Erzeugt einen XML Eintrag für ein einzelnes Daten Mitglied (PropertyName) in dem XML Dokument
 - XMLNodeReference enthält X-NodeRef auf einen den Instanz Knoten in dem X-Dokument
 - Mehrere Implementierungen in XMLSerializable um "jeden" Datentyp (incl. IXMLSerializable für Klassen)

Objektorientiert auf dem AppServer 40

Consultingwerk
software architecture and development

Kaskadierende Serialisierung

- Eine SUPER Klasse (die Klasse, von der die *Invoice* Klasse abgeleitet wird) kann ebenfalls Daten Mitglieder enthalten, die serialisiert werden müssen
 - Gegen Sie der SUPER Klasse die Möglichkeit Ihre Daten Mitglieder zu *Serialize()*en
- Daten Mitglieder können von Klassen Datentypen sein, so dass diese ebenfalls serialisiert werden müssen (incl. deren SUPER Klassen)

Objektorientiert auf dem AppServer 41

Consultingwerk
software architecture and development

Kaskadierende Serialisierung, Beispiel

- *Invoice* Klasse erbt vom *SalesDocument*
 - *SalesDocument* hat die folgenden Mitglieder:
 - *DocumentNumber* AS INTEGER
 - *DocumentAddress* AS *AddressInformation*
 - *Invoice* selbst hat die folgenden Mitglieder:
 - Buffer *bInvoice* (Table sports2000.invoice)
 - Buffer wird über Hilfsklasse serialisiert, über TABLE, DBNAME, ROWID und einen SHA1 Hash des aktuellen Werts

Objektorientiert auf dem AppServer 42

Consultingwerk
software architecture and development

Kaskadierende Serialisierung, Beispiel

```
<?xml version="1.0" ?>  
<Invoice>  
  <bInvoice DBNAME="sports2000" />  
  ROWID="0x000000000000006f"  
  SHA1="NVWqfsIOTW8IXfAoU9ZhgOwJIm8="  
  TABLE="Invoice" />  
  <DocumentNumber>2</DocumentNumber>  
  <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>
```

Objektorientiert auf dem AppServer

43

Consultingwerk
software architecture and development

Deserialisierung

- Prozess aus der Information in dem XML Dokument wieder eine laufende Instanz zu erzeugen
- Entweder über einen Konstruktor oder einen Methodenaufruf
 - Konstruktor erzeugt immer eine neue Instanz
 - Mehrere Konstruktoren sind möglich
 - Methodenaufruf kann eine bereits laufende Instanz eines vorhergehenden AppServer requests wieder verwenden

Objektorientiert auf dem AppServer

44

Consultingwerk
software architecture and development

Deserialisierung

- Beachten Sie ebenfalls kaskadierende Dokumente
- Deserialisieren Sie Klassen Typen ebenso wie die Mitglieder der SUPER Klasse
- Loopen Sie durch die direkten Child-Knoten des aktuellen Instanz-Knotens und besetzen Sie die Daten Mitglieder (siehe CASE Statement im Beispiel)
- Sollten Knoten Klassen Typen repräsentieren, nutzen Sie deren Deserialisierungsmethode

Objektorientiert auf dem AppServer 45

Consultingwerk
software architecture and development

Serialisierung/Deserialisierung Beispiele

- Speichern der SessionInformation im AppServer Session Context
- Übergabe der Invoice Klasse zwischen Client und AppServer zwischen Methodenaufrufen

Objektorientiert auf dem AppServer 46

Consultingwerk
software architecture and development

Deserialisierung, sample

```
DO i = 1 TO hObjectNode:NUM-CHILDREN :  
    hObjectNode:GET-CHILD (hNode, i) .  
  
    IF hObjectNode:NUM-CHILDREN <> 1 THEN NEXT.  
  
    hObjectNode:GET-CHILD(hChild, 1) .  
  
    CASE hObjectNode:NAME:  
        WHEN „DocumentNumber" THEN  
            ASSIGN iDocumentNumber  
                = hObjectNode:GET-CHILD(hChild, 1):NODE-VALUE .  
        WHEN „DocumentAddress" THEN DO:  
            ASSIGN documentAddress = NEW  
                DocumentAddress().  
            documentAddress:Deserialize(hObjectNode) .  
    END CASE.
```


Objektorientiert auf dem AppServer 47

Consultingwerk
software architecture and development

Beachten Sie die Grundlagen der ABL!

- Ein Client der keine Verbindung zur Datenbank besitzt, kann eine Klasse nicht instanzieren, die auf Datenbanktabellen zugreift
– RCODE-INFO:DB-REFERENCES
- Transaktionen und Record-Locks sind an die Client Sitzung gebunden und können nicht serialisiert werden
- Wenn Sie einen Datensatz wieder lesen, kann dieser gelockt oder bereits geändert sein


Objektorientiert auf dem AppServer 48



Agenda

- Objektorientierung in der ABL
- Verwendung von Klassen auf dem AppServer
- Proxy Prozeduren
- Objektpersistenz auf dem AppServer
- Serialisierung
- Objektaustausch mit nicht OpenEdge Anwendungen

Objektorientiert auf dem AppServer 49



XML Serialisierung mit .NET Proxies

- XMLSerializer class in C#
- Erzeugen einer C# Klasse die der Invoice Klasse entspricht

Objektorientiert auf dem AppServer 50

Zusammenfassung

- Objektorientierung ist in der ABL verfügbar
- OO in ABL bietet gute Kontrolle über den Code und die Ausführung (strong typing, Vererbung, Interfaces, private/protected/public Zugriff)
- Verwendung von Klassen auf dem AppServer ist ebenfalls möglich
- Serialisierung kann implementiert werden
- Auch wenn einige Features noch fehlen (Reflektion, native Serialisierung) gibt es keinen Grund nicht los zu legen
- Entwerfen Sie Ihre Klassen unter dem Gesichtspunkt kommender Erweiterungen der OOABL