Consultingwerk

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany, subsidiaries in UK and Romania
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration
- Experts in OpenEdge Application Modernization

Mike Fechner

- Director, Lead Modernization Architect and Product Manager of the SmartComponent Library and WinKit
- Specialized on object oriented design, software architecture, desktop user interfaces and web technologies
- 30 years of Progress experience (V5 … OE12)
- Active member of the OpenEdge community
- Frequent speaker at OpenEdge related conferences around the world

SmartComponent Library Entwicklerframework

- Unterstützt das Sichern der Investitionen in ihre OpenEdge basierenden Anwendungen
- Framework wurde entworfen sowohl um bestehende OpenEdge Anwendungen zu modernisieren als auch um eine solide und zukunftssichere Grundlage für neue Anwendungen bereit zu stellen
- In der Cloud und on-premise
- OERA, CCS
- Die Architektur des Frameworks ist auf Integration mit kommenden Technologien und die Umsetzung neuer Business Requirements ausgelegt

User Interface Flexibilität

- Windows Desktop User Interfaces mit .NET
- Angular Web Anwendungen (Kendo UI)
- Mobile Anwendungen (NativeScript)
- Offene Schnittstellen (z.B. RESTful)
- Partner User Interfaces (z.B. AKIOMA)

Kendo UI
THE ART OF WEB DEVELOPMENT

Agenda

- **Modernisierung von OpenEdge Anwendungen**
- Quellcodeanalyse
- Proparse
- Use-Case 1: UI Layout überführen
- Use-Case 2: Validierungscode erkennen und migrieren
- Use-Case 3: Erkennen von Call-Pfaden
- Schlussfolgerung

Modernisierungsziele

- Funktionale Anforderungen
  - Hinzufügen neuer Fach-Module
  - Verändern der bestehenden Fach-Logik
- Nichtfunktionale Anforderungen
  - Neues User-Interface / User-Experience
    - Moderner Desktop
    - Web
    - Mobile
  - Neue Architektur
  - Performance (z.B. im WAN), vermeiden von Citrix

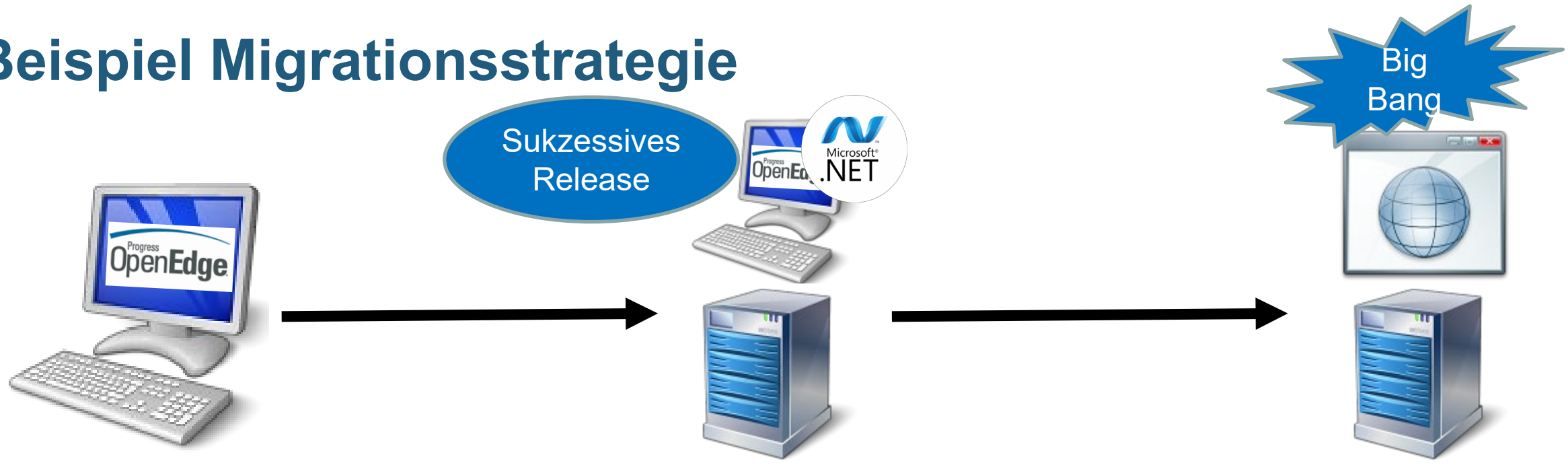
Was macht die Anwendung

Wie löst die Anwendung die Anforderungen

Modernisierungsstrategien

- Modernisierung der vollständigen Anwendung
  - Von TTY oder ABL GUI zu modernem Desktop, Web oder Mobile
  - Entscheidung über die „finale“ UI Technologie
  - Muss es immer ein Web Frontend sein?
  - Direkt vom Status-Quo zur finalen Lösung – oder über Zwischenschritt
- Umsetzen erster neuer Features in einer neuen Technologie
  - Mobiler Client für Teile der Anwendung
  - REST/RESTful Schnittstellen für Teile der Anwendungslogik
  - Reduzieren des Risikos
  - Erfahrung mit neuer Technologie sammeln

Beispiel Migrationsstrategie



Logistikanwendung Progress Dynamics 75% der Funktionen in 2 Treeview-Masken

Modernes Desktop UI, PASOE Backend, SmartComponent Library

Moderne Web-Anwendung, PASOE Backend, SmartComponent Library

Agenda

- Modernisierung von OpenEdge Anwendungen
- **Quellcodeanalyse**
- Proparse
- Use-Case 1: UI Layout überführen
- Use-Case 2: Validierungscode erkennen und migrieren
- Use-Case 3: Erkennen von Call-Pfaden
- Schlussfolgerung

Quellcode Analyse

- Quellcode bestehender Anwendungen in der Regel die einzige vollständige Beschreibung der Funktionalität
- Analysierbarkeit von Quellcode Grundlage für jede (teil-)automatisierte Migration oder Bewertung von Code
- Statische Code Analyse: Anhand von Quellcode, ohne Ausführung
- Laufzeitanalyse in der Runtime

Statische Codeanalyse mit OpenEdge Bordmitteln

- Über Compiler-Features
- Cross-Reference (XREF oder XML-XREF)
  - Include Files, RUN
  - Buffer/Variablen Zugriff
  - Strings (für Translation)
- LISTING
  - Buffer- und Transaction Scope
  - Frame Scoping
- PREPROCESS oder DEBUG-LISTING

Lautzeitanalyse

- PROFILER
  - Aufgerufene Routinen und Timing
  - Aufgerufene Statements und Timing
- Client logfile (LOG-MANAGER System Handle)
  - Queries
  - aufgerufene Routinen mit Parametern und Rückgabe
  - Timings
- Dynamic inspection, Widget-Tree oder Queries
  - erfordert Zugriff auf ausgeführte Anwendung, teilweise Quellcode-Veränderung

Agenda

- Modernisierung von OpenEdge Anwendungen
- Quellcodeanalyse
- **Proparse**
- Use-Case 1: UI Layout überführen
- Use-Case 2: Validierungscode erkennen und migrieren
- Use-Case 3: Erkennen von Call-Pfaden
- Schlussfolgerung

Proparse

- Werkzeug um einen abstrakten Syntax-Baum für ABL (4GL) Code zu erstellen – AST (abstract syntax tree)
- Statische Code Analyse (Code muss nicht ausgeführt werden)
- Interpretation der Grammatik und Semantik der ABL
- Verständnis gültiger Keywords und gültiger Kombinationen sowie deren Bedeutung (z.B. RUN … PERSISTENT SET hProc)
- Verständnis von Sprachkonzepten wie Scopes
- Verständnis bzw. Reimplementierung des ABL Preprocessors
  - &IF WINDOW-SYSTEM NE TTY
  - Include Files und Parameter

Proparse

- In der Lage jeden ABL Code zu interpretieren, der sich kompilieren lässt
- Vergleichbare Anforderungen wie Compiler
  - DB-Schema
  - PROPATH
  - vollständiger referenzierter Quellcode (z.B. Include Files)
  - unverschlüsselter Quellcode

Warum ein abstrakter Syntax-Baum

- Analyse von ABL Quellcode als Text-Dateien kompliziert und i.d.R. nur rudimentär

- AST abstrahiert Sicht auf Programmstruktur von dem ursprünglichen Quellcode

ABL Syntax flexibel …

- Formatierung, Zeilenumbrüche
- Große Menge an Keywords mit relevanter Bedeutung
- Abgekürzte Keywords
- Reihenfolge von Keywords teilweise relevant
- Groß-/Kleinschreibung von Keywords
- Einige Keywords können als Identifier genutzt werden (Variablenname)
- Hochkommata bei String-Literalen, einfach, doppelt, maskiert, in Kombination
- Kommentare, geschachtelt, …

Darum ....

simple-1.p | simple-2.p

```
/*------------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */

DEFINE VARIABLE i AS INTEGER NO-UNDO INITIAL 21 .

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

MESSAGE i * 2
    VIEW-AS ALERT-BOX.
```

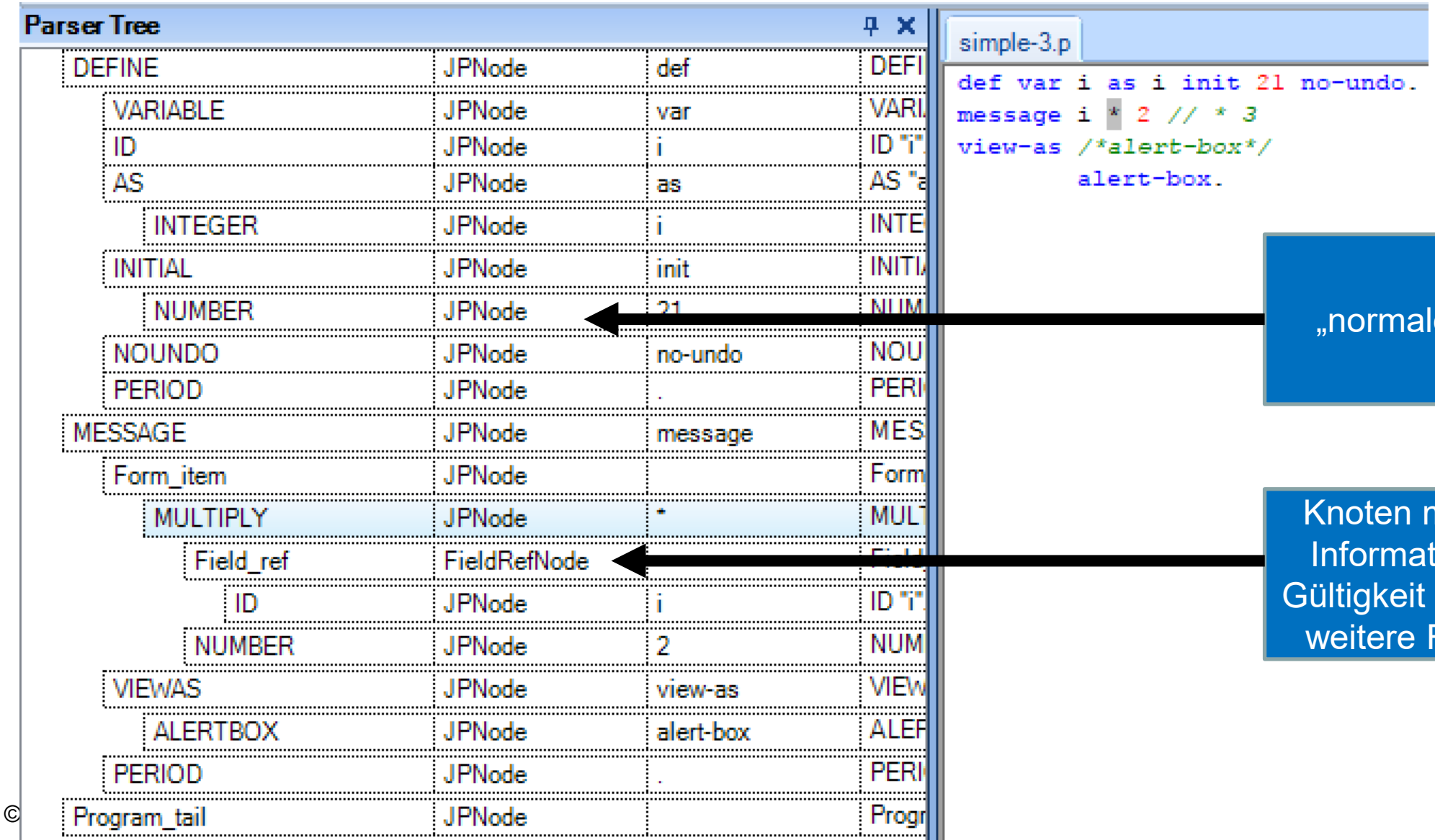
simple-1.p | simple-2.p

```
def var i as i init 21 no-undo. message i * 2 view-as alert-box.
```

simple-1.p | simple-2.p | simple-3.p

```
def var i as i init 21 no-undo.
message i * 2 // * 3
view-as /*alert-box*/
        alert-box.
```

AST und Progress Compiler sollten in allen drei Fällen das identische Resultat liefern



„normaler Knoten“

Knoten mit weiteren Informationen, z.B. Gültigkeit der Variable, weitere Referenzen

Proparse

- Ursprünglicher Autor: John Green / Joanju
- http://www.joanju.com/proparse/
- http://www.oehive.org/proparse
- http://www.joanju.com/analyst/javadoc/index.html
- Eclipse public license (EPL2, GPL3)
- Ermittelt den *Abstract Syntax Tree* auf Basis einer Compile-Unit
- Ersetzt nicht den Compiler oder Syntax-Checker
  - vergleichbare Anforderungen
- Basiert auf **ANTLR** 2.7 (aktuell ist Version 4)

Proparse

- Mehrere öffentliche Quellcode-Repositories
  - github.com/oehive/proparse
  - **github.com/consultingwerk/proparse**
  - github.com/Riverside-Software/sonar-openedge/tree/develop/proparse

- Nach extrem ruhiger Phase zwischen 2000 und 2010 wird Proparse nun wieder aktiv gepflegt
- Support für vollständige OpenEdge 12.1 Syntax verfügbar, OpenEdge 12.2 und 12.3 in Arbeit, z.B. var Statement und =+ Operatoren

ANTLR 2.7

- "**An**other **T**ool for **L**anguage **R**ecognition"
- Toolkit zum Entwickeln von Sprach-Parsern
- Erstellte Parser (incl. Proparse) sind Java Code
- Aktuellere Versionen von ANTLR bieten wesentlich mehr Tool-Unterstützung an
  - incl. Generierung von Parsern in weiteren Technologien
- Proparse enthält große Menge an weiterem Java-Code
- Sofern man nicht vor hat, Proparse zu pflegen, sollte das aber nicht relevant sein

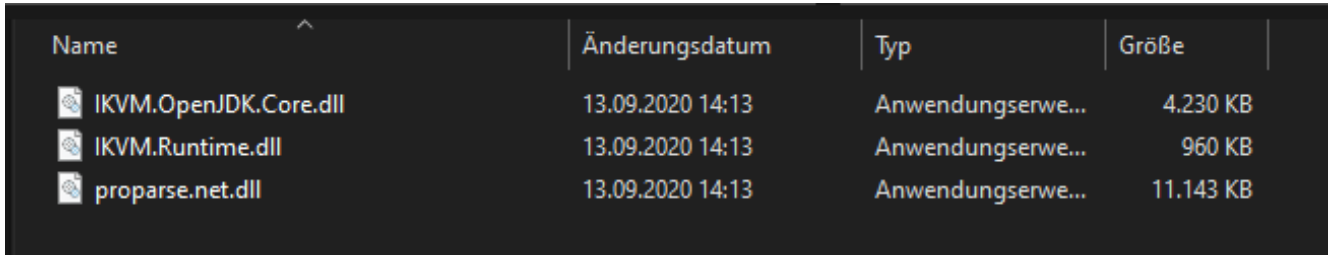
Beispiel Proparse Grammatik

```
messagestate
            :           MESSAGE^
                        (color_anyorvalue)?
                        (message_item)*
                        (message_opt)*
                        (in_window_expr)?
                        state_end
                        {sthd(##,0);}
            ;
message_item
            :           (           skipphrase
                        |           expression
                        )
                        {##=#([Form_item],##);}
            ;
```

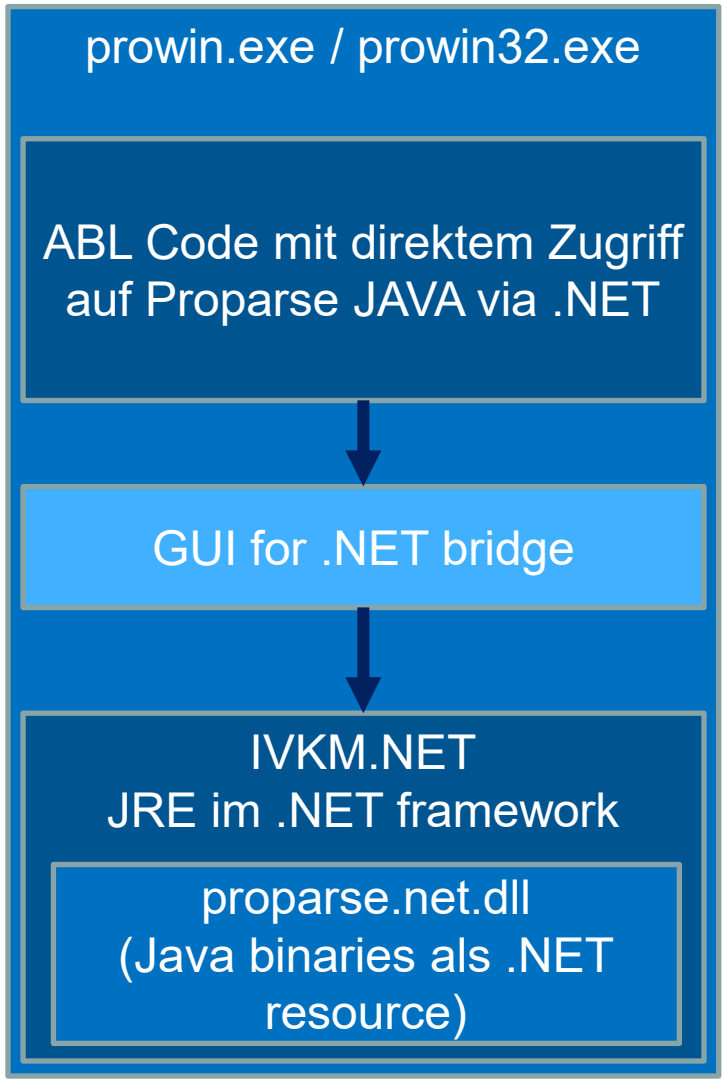
```
message_opt
            :           VIEWAS^ ALERTBOX
                        (MESSAGE|QUESTION|INFORMATION|ERROR|WARNING)?
                        (           (BUTTONS | b:BUTTON {#b.setType(BUTTONS);})
                                    (YESNO|YESNOCANCEL|OK|OKCANCEL|RETRYCANCEL)
                        )?
                        (title_expr)?
            |           SET^ field ({LA(2)!=ALERTBOX}? (options{greedy=true;}: formatphrase)? |)
            |           UPDATE^ field ({LA(2)!=ALERTBOX}? (options{greedy=true;}: formatphrase)?
|)
            ;
```

Proparse in ABL?

- Proparse.NET
- IKVM Java Runtime
- Spinn off des Mono-Projects
- https://www.windwardstudios.com/blog/ikvm-is-alive-well



Name	Änderungsdatum	Typ	Größe
IKVM.OpenJDK.Core.dll	13.09.2020 14:13	Anwendungserwe...	4.230 KB
IKVM.Runtime.dll	13.09.2020 14:13	Anwendungserwe...	960 KB
proparse.net.dll	13.09.2020 14:13	Anwendungserwe...	11.143 KB



Demo

- ABL Quellcode zum rekursiven Auswerten eines ABL Programmes

Einsatzfelder für Proparse in der Migration

- Analyse von ggf. Refactoring von jedweden ABL Quellcode
- Framework-Migration
  - ADM/ADM2
  - Progress Dynamics
  - DWP
  - …
- Home-grown Frameworks oder Template-Systeme
- Code ohne Unterstützung von Frameworks

Tools auf Basis von Proparse

- CABL / SonarCube Plugin für OpenEdge (Riverside Software)
- Business Entity Designer der SmartComponent Library für round-trip Code Generatoren
- ABL Source Code Migrations-Utilities
- Call-Graph Analyse
- Navigation in unseren Quellcode Editoren

- Traditionell auf Basis von Proparse.NET
- Neu: Proparse.Web: Tomcat Web Application auf Basis von Proparse zur Analyse von Quellcode in PASOE / Docker

Agenda

- Modernisierung von OpenEdge Anwendungen
- Quellcodeanalyse
- Proparse
- **Use-Case 1: UI Layout überführen**
- Use-Case 2: Validierungscode erkennen und migrieren
- Use-Case 3: Erkennen von Call-Pfaden
- Schlussfolgerung

ABL UI Layout überführen

- Wenn das Ziel einer Modernisierung primär die Überführung zu einer neuen Architektur oder UI-Technologie ist, kann das bestehende ABL UI Layout ein wertvoller Startpunkt sein
  - Anordnung von Feldern
  - Datenbindung
  - UI Eventlogik
- späteres Nachbearbeiten nicht ausgeschlossen

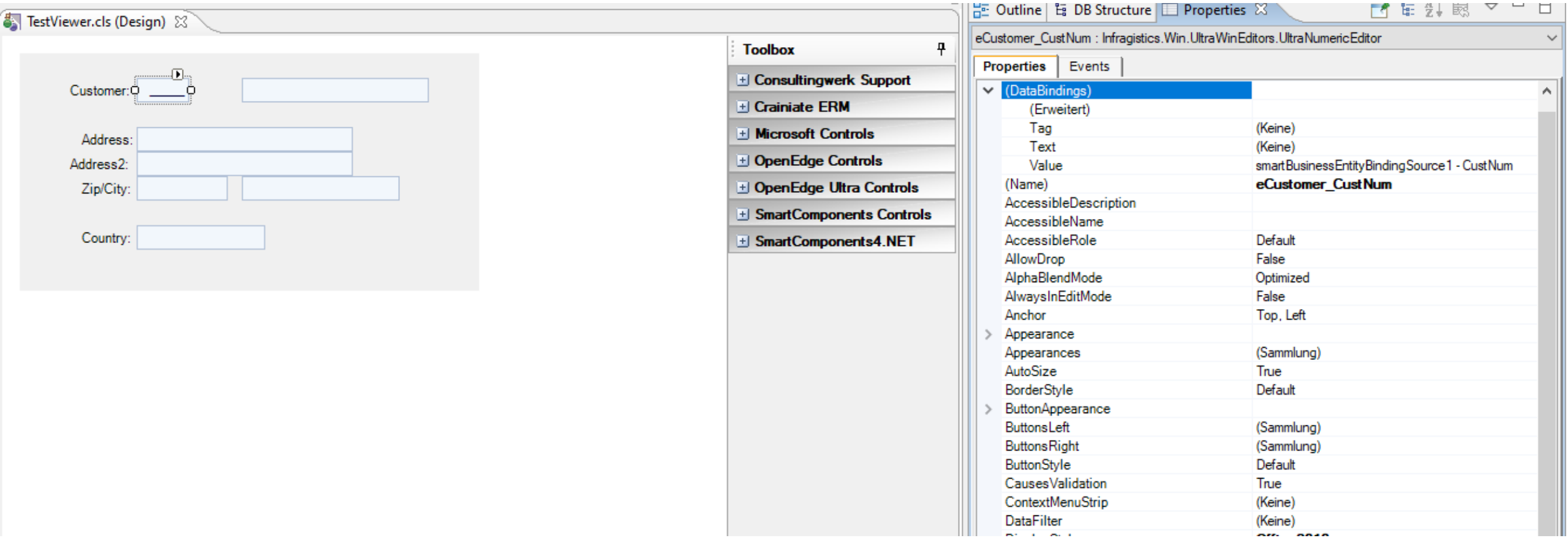
ABL UI Layout überführen

- FORM Statements
- DEFINE FRAME Statements
- Mehrere Statements können kombiniert werden
- Feldeigenschaften
  - explizit definiert
  - aus dem Data-Dictionary geerbt
  - Default-Values für verschiedenen Controls
- VIEW-AS Phrase
- Positionen explizit (ROW/COL) oder implizit durch Reihenfolge, SKIP

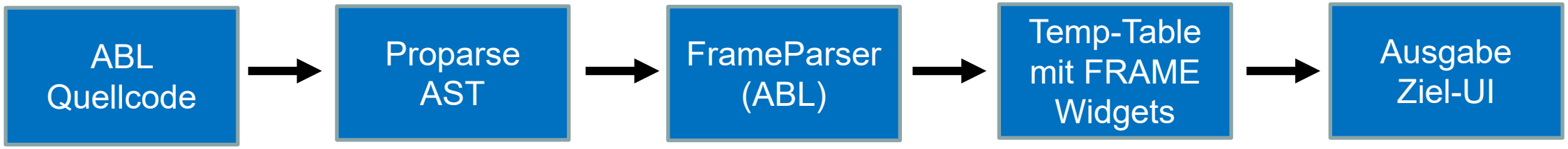
Beispiel ABL Frame-Layout

```
FORM Customer.CustNum         AT ROW 1 COL 20 LABEL "Customer":T
     Customer.Name            AT ROW 1 COL 38 NO-LABEL
     Customer.Address         AT ROW 3 COL 20
     Customer.Address2        AT ROW 4 COL 20
     Customer.PostalCode      AT ROW 5 COL 20 LABEL "Zip/City":T
     Customer.City            AT ROW 5 COL 38 NO-LABEL
     Customer.Country         AT ROW 7 COL 20
     WITH FRAME customerFrame .

FORM Salesrep.SalesRep SKIP
     Salesrep.RepName SKIP
     Salesrep.Region
     WITH FRAME salesrepFrame .
```

Migration zu GUI for .NET



Analyse von Frames



- .w oder .p
- Procedure Editor, AppBuilder oder sonstiges
- Mehrere FRAME‘s
- ADM / ADM2
- Framework / ohne Framework

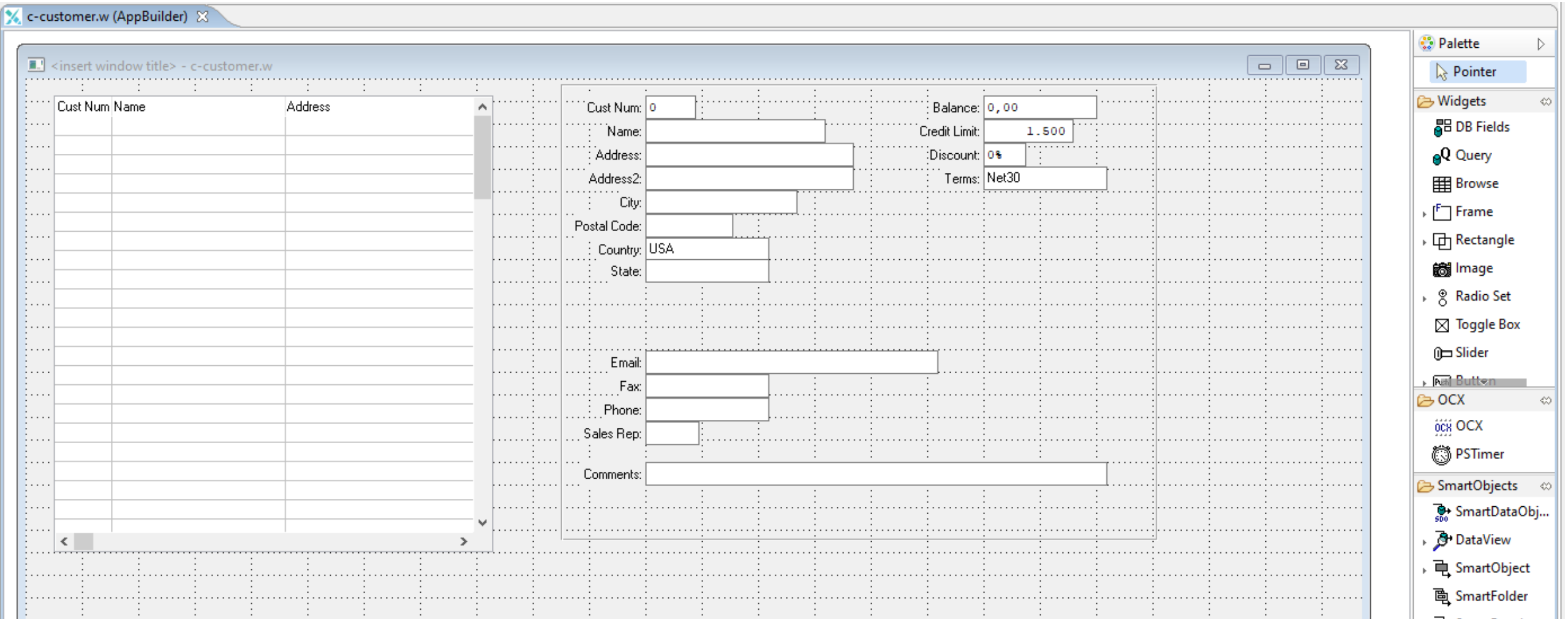
- **ABL Klasse**, Teil unseres Toolkits, empirisch verbessert
- Interpretation AST
- Ergänzung durch ABL-Defaults oder Data Dictionary Eigenschaften
- **Anpassbarkeit** durch Callbacks und Events

- GUI for .NET
- Angular Web UI
- JSON Markup
- User Interface Repository

Migration von ABL UI Layout

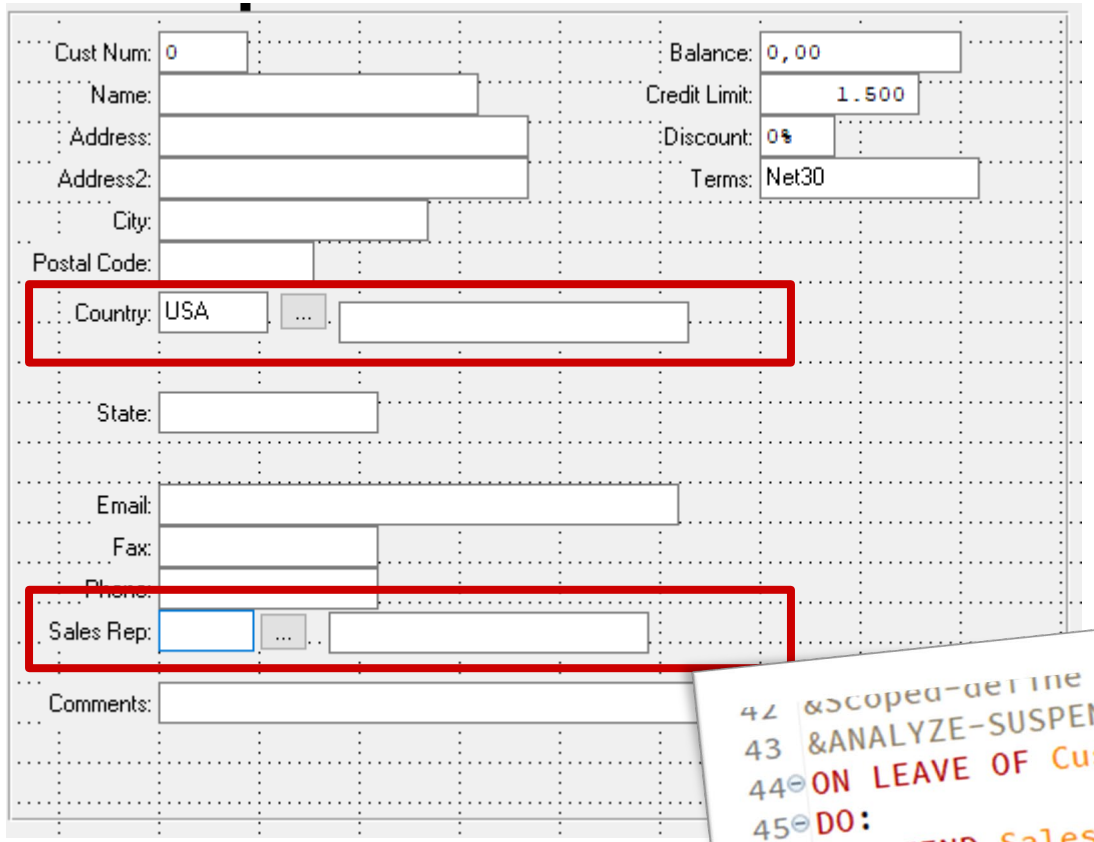
- Abkoppeln von Datenbankfeldern
  - Datenbindung mach Migration i.d.R. zu Business Entity
  - Möglichkeit zur Erstellung einer Business Entity
- Kenntnis der Ziel UI Struktur verbessert Resultat
  - z.B. Ignorieren von BROWSE-Widgets, da diese i.d.R. separat realisiert werden
- Trigger Code kann übernommen und migriert werden
- Anpassbarkeit durch Callbacks und Events
  - Manipulation von Widget-Properties
  - Ignorieren bestimmter Widgets (z.B. RECTANGLE-Overflow)

Demo ABL Frame Migration



Erkennen von Lookups oder Zoom-Feldern

- Beispiel für Adaption von UI-Pattern
- Übliche Kombination in Erfassungsmasken
  - Feld für Schlüsselfeld (z.B. Salesrep-Code), Details (z.B. Salesrep-Name)
  - Lookup-Button zum Öffnen eines Suchdialoges
  - Event Handler für
    - Shortcut Key zum Öffnen des Lookup Dialoges
    - Choose-Trigger des Lookup Buttons
    - Leave des Lookup Feldes
- Oft umfangreich in der ABL Implementiert
- In der Regel bieten neue UI Frameworks eine Lookup Komponente




```
42  &Scoped-define SELF-NAME Customer.SalesRep
43  &ANALYZE-SUSPEND _UIB-CODE-BLOCK _CONTROL Customer.SalesRep C-Win
44  ON LEAVE OF Customer.SalesRep IN FRAME DEFAULT-FRAME /* Sales Rep */
45  DO:
46      FIND Salesrep WHERE Salesrep.Salesrep = Customer.Salesrep:SCREEN-VALUE
47          NO-LOCK NO-ERROR .
48
49      IF AVAILABLE Salesrep THEN
50          DISPLAY Salesrep.Salesrep @ Customer.Salesrep
51                  Salesrep.RepName WITH FRAME {&frame-name}.
52  END
```

Erkennen von Lookups oder Zoom-Feldern

- Frame-Parser erweiterbar durch Prozessoren für UI Patterns
- „DetectLookupControlFrameMigrationPreProcessor“
- Analyse von Positionen von Widgets in der Ursprungsmaske
- Analyse von Trigger-Code, z.B. LEAVE
  - FIND Statement
  - DISPLAY Statements
  - …
- Anpassbarkeit an eigenen Coding-Style

Demo

- Migration Customer Viewer mit Lookup Feldern

Parsen des LEAVE Triggers (Auszug)

```
208         oWalker = NEW NodeWalker ("FIND":U) .
209         oFindChildNodeAction = NEW FindFirstMatchingChildNodeAction() .
210         oWalker:WalkNodes(poNode, oFindChildNodeAction) .
211
212         IF NOT VALID-OBJECT (oFindChildNodeAction:JPNode) THEN
213             RETURN ? .
214
215         oFindStatement = oFindChildNodeAction:JPNode .
216
217         oRecordName = CAST (ProparseHelper:FindChildNodeOfNodeType (oFindStatement, "RECORD_NAME":U), RecordNameNode) .
218
219         IF NOT VALID-OBJECT (oRecordName) THEN
220             RETURN ? .
221
222         ASSIGN cLookupTable = ProparseHelper:FullyQualifiedTableName (oRecordName)
223                oWhere       = ProparseHelper:FindChildNodeOfNodeType(oRecordName, "WHERE":U)
224                oNodeWrapper = NEW JPNodesWrapper(oWhere, "EQ":U).
225
226         {Consultingwerk/foreachABL.i JPNode oEQ in oNodeWrapper '' eqLoop}
227
228             IF NEW BufferFieldName (ProparseHelper:GetFullFieldName(oEq:firstChild():nextSibling())):WithTableName() = pcFieldName THEN DO:
229                 ASSIGN cLookupKeyField = ProparseHelper:GetFullFieldName(oEq:firstChild()) .
230
231                 LEAVE eqLoop  .
232             END.
233         END.
```

Migration von ABL UI Layout

- Vergleichbare Migrations-Routinen für weitere ABL Widgets
  - BROWSER
  - Vollständige Fenster etc.
- Trigger Code kann auch als „Skeleton“ in nicht ABL Quellcode ausgegeben werden
  - JavaScript / TypeScript
  - In Zukunft: Rewrite von simpler UI Logik geplant
    - z.B. ABL zu TypeScript/Angular
    - z.B. IF THEN auf Basis von Feldinhalten bei LEAVE oder VALUE-CHANGED

Agenda

- Modernisierung von OpenEdge Anwendungen
- Quellcodeanalyse
- Proparse
- Use-Case 1: UI Layout überführen
- **Use-Case 2: Validierungscode erkennen und migrieren**
- Use-Case 3: Erkennen von Call-Pfaden
- Schlussfolgerung

Validierungscode

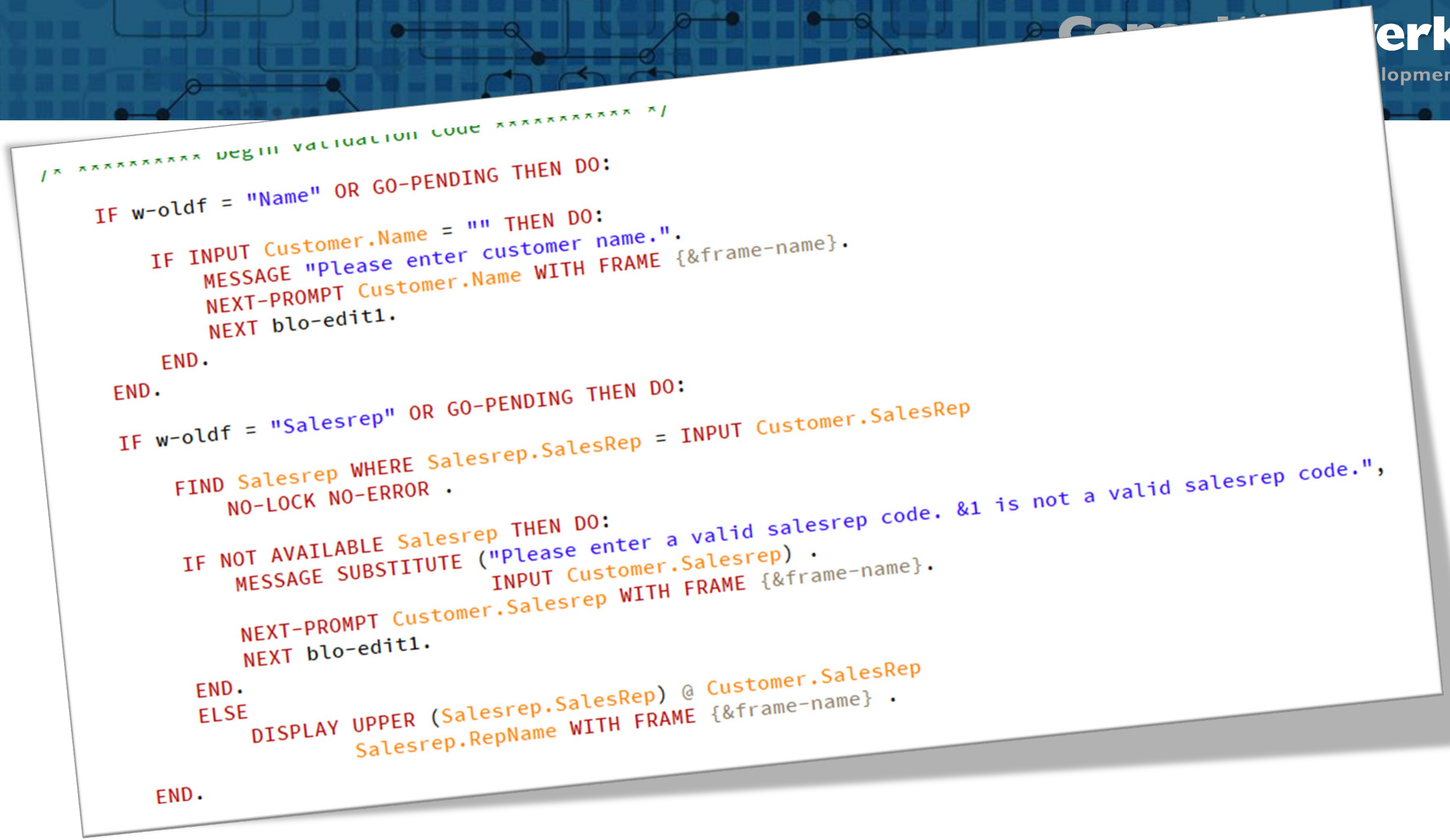
- Validierung von (Benutzer-)Eingaben Kernfunktion jeder Business Applikation
- Implementierung in UI oder bereits getrennt
- LEAVE oder VALUE-CHANGED Trigger
- Save Button
- Interne Prozeduren/Funktionen, z.B. ValidateCustNum
- ADM2 SmartDataObject
  - Migration von Validierungscode im SDO
  - Ausführen von Validierungscode in externer Data-Logic Procedure

Struktur von Validierungscode (vereinfacht)

- Prüfung: IF THEN ELSE
- ggf. direkter Zugriff auf Maskenfelder
- ggf. Zugriff auf Datenbank für Schlüsseltabellen
- Bei Validierungsfehlern wird der User informiert (z.B. MESSAGE)
- Ggf. Eingabefokus auf das verursachende Feld setzen
- Speichern wird abgebrochen / nicht ausgeführt

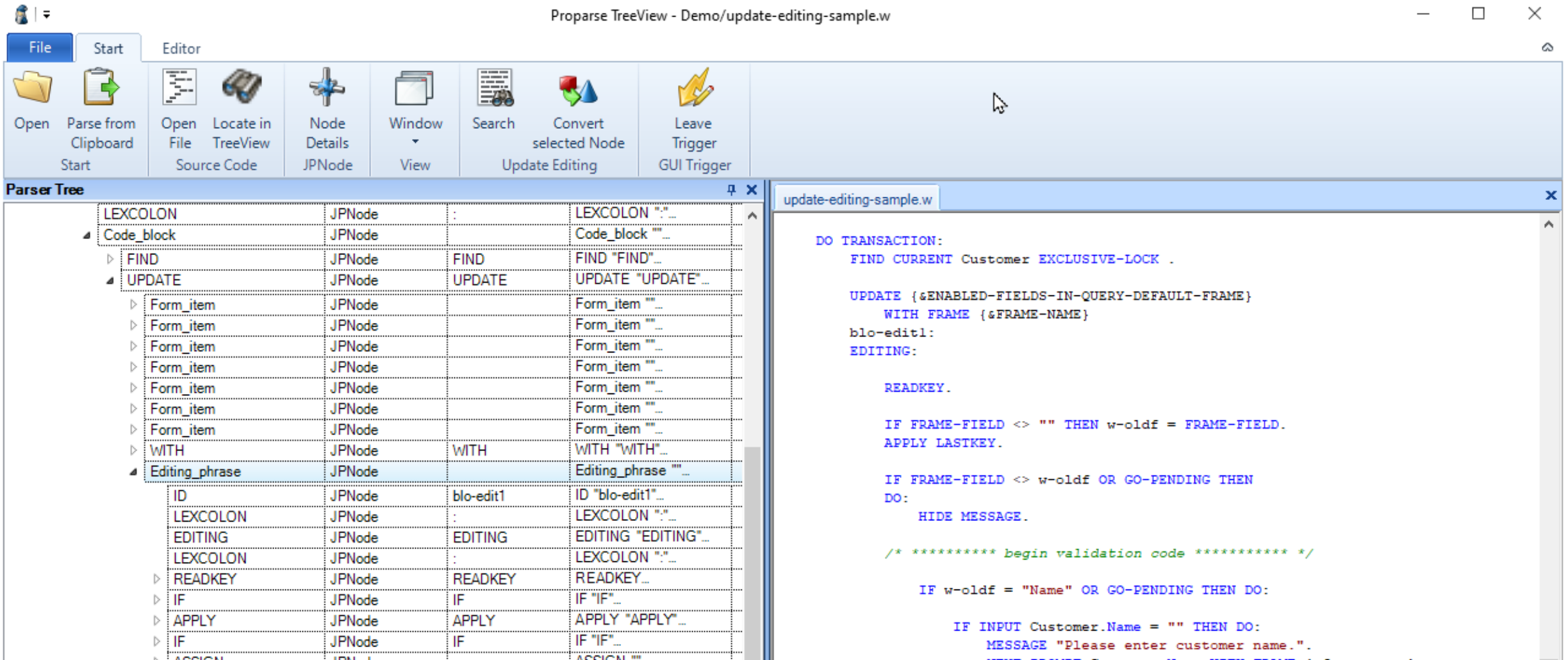
Beispiel UPDATE EDITING

- In TTY Code immer noch übliche Form der Validierung
- Zeichenweise Validierung von Maskencode oder „LEAVE“ bei Wechsel des FRAME-FIELD‘s
- Eher seltener in GUI Code
- Typische Element
  - UPDATE Statement
  - FRAME-FIELD Funktion
  - GO-PENDING Funktion
  - NEXT-PROMPT Statement



```
/* ********** Begin validation code ********** */

    IF w-oldf = "Name" OR GO-PENDING THEN DO:

        IF INPUT Customer.Name = "" THEN DO:
            MESSAGE "Please enter customer name.".
            NEXT-PROMPT Customer.Name WITH FRAME {&frame-name}.
            NEXT blo-edit1.
        END.
    END.

    IF w-oldf = "Salesrep" OR GO-PENDING THEN DO:

        FIND Salesrep WHERE Salesrep.SalesRep = INPUT Customer.SalesRep
            NO-LOCK NO-ERROR .

        IF NOT AVAILABLE Salesrep THEN DO:
            MESSAGE SUBSTITUTE ("Please enter a valid salesrep code. &1 is not a valid salesrep code.",
                                INPUT Customer.Salesrep) .

            NEXT-PROMPT Customer.Salesrep WITH FRAME {&frame-name}.
            NEXT blo-edit1.
        END.
        ELSE
            DISPLAY UPPER (Salesrep.SalesRep) @ Customer.SalesRep
                    Salesrep.RepName WITH FRAME {&frame-name} .

    END.
```

Demo Migration UPDATE EDITING Block



```
FIND Salesrep WHERE Salesrep.SalesRep = INPUT Customer.SalesRep
    NO-LOCK NO-ERROR .

IF NOT AVAILABLE Salesrep THEN DO:
    MESSAGE SUBSTITUTE ("Please enter a valid salesrep code. &1 is not a valid salesrep code."
                        INPUT Customer.Salesrep) .
    NEXT-PROMPT Customer.Salesrep WITH FRAME {&frame-name}.
    NEXT blo-edit1.
END.
ELSE DO:
    DISPLAY UPPER (Salesrep.SalesRep) @ Customer.SalesRep
            Salesrep.RepName WITH FRAME {&frame-name} .
```

```
FIND Salesrep WHERE Salesrep.SalesRep = eCustomer.SalesRep
    NO-LOCK NO-ERROR .

IF NOT AVAILABLE Salesrep THEN DO:
    Consultingwerk.Util.DatasetHelper:AddErrorString (BUFFER eCustomer:HANDLE,
                                                      SUBSTITUTE ("Please enter a valid salesrep code. &1 is not a valid salesrep code.",
                                                                  eCustomer.SalesRep),
                                                      "SalesRep":U) .
END.
ELSE DO:
    ASSIGN eCustomer.SalesRep = UPPER (Salesrep.SalesRep)
           eCustomer.RepName = Salesrep.RepName .
```

Migration UPDATE EDITING Code

- Von TTY UI hin zu Validierung für Business Entity (OERA bzw. CCS)
- Kontrollstrukturen im Code bleiben (weitestgehend) erhalten
- Zugriff auf UI Controls wird ersetzt durch Zugriff auf Temp-Table der Business Entity
- Benutzerinteraktion (MESSAGE, PROMPT-FOR, …) wird ersetzt durch API‘s des Backend-Frameworks
- Migration des Zugriffes auf Datenbanktabellen kann durch Zugriff auf weitere Business Entities ersetzt werden

Agenda

- Modernisierung von OpenEdge Anwendungen
- Quellcodeanalyse
- Proparse
- Use-Case 1: UI Layout überführen
- Use-Case 2: Validierungscode erkennen und migrieren
- **Use-Case 3: Erkennen von Call-Pfaden**
- Schlussfolgerung

Erkennen von Call-Pfaden

- Erkennen von Call-Pfaden ist die Basis für Unterstützung bei der Aufsplittung monolithischer Anwendungen in Front- und Backend
- Einzelne Routinen können auf Abhängigkeit zur Datenbank (read-only und Transaktional) untersucht werden
  - ganzes .p oder .w File
  - Interne Prozeduren oder Funktionen
  - Klassen und deren Methoden
- Wechsel im Call-Pfad von UI zu Datenbank ist ein Indiz für die Notwendigkeit Code Aufzuteilen

Call Objekt in Proparse

- Proparse löst bei Aufrufen in Prozeduralen Code das RUN Statement auf
- Details über den Aufruf werden in einem Call Objekt dargestellt
- RUN <externe procedure>
- RUN <interne procedure>
- RUN <externe procedure> PERSISTENT SET <handle-var>
- RUN <interne procedure> IN <handle-var>

org.prorefactor.treeparser

Class Call

java.lang.Object
  org.prorefactor.treeparser.SemanticRecord
    org.prorefactor.treeparser.Call

```
public class Call
extends SemanticRecord
```

Represents a Call to some 4GL procedure. The target procedure is identified by the external and internal procedure names. The expecte values for externalName and internalName are as follows:

```
                              externalName - internalName
 run  [in this-procedure]: compile-unit
 run  in .       : handle:target
 run  [persistent [...]. : compile-unit   null
```

Author:

pcd

Constructor Summary

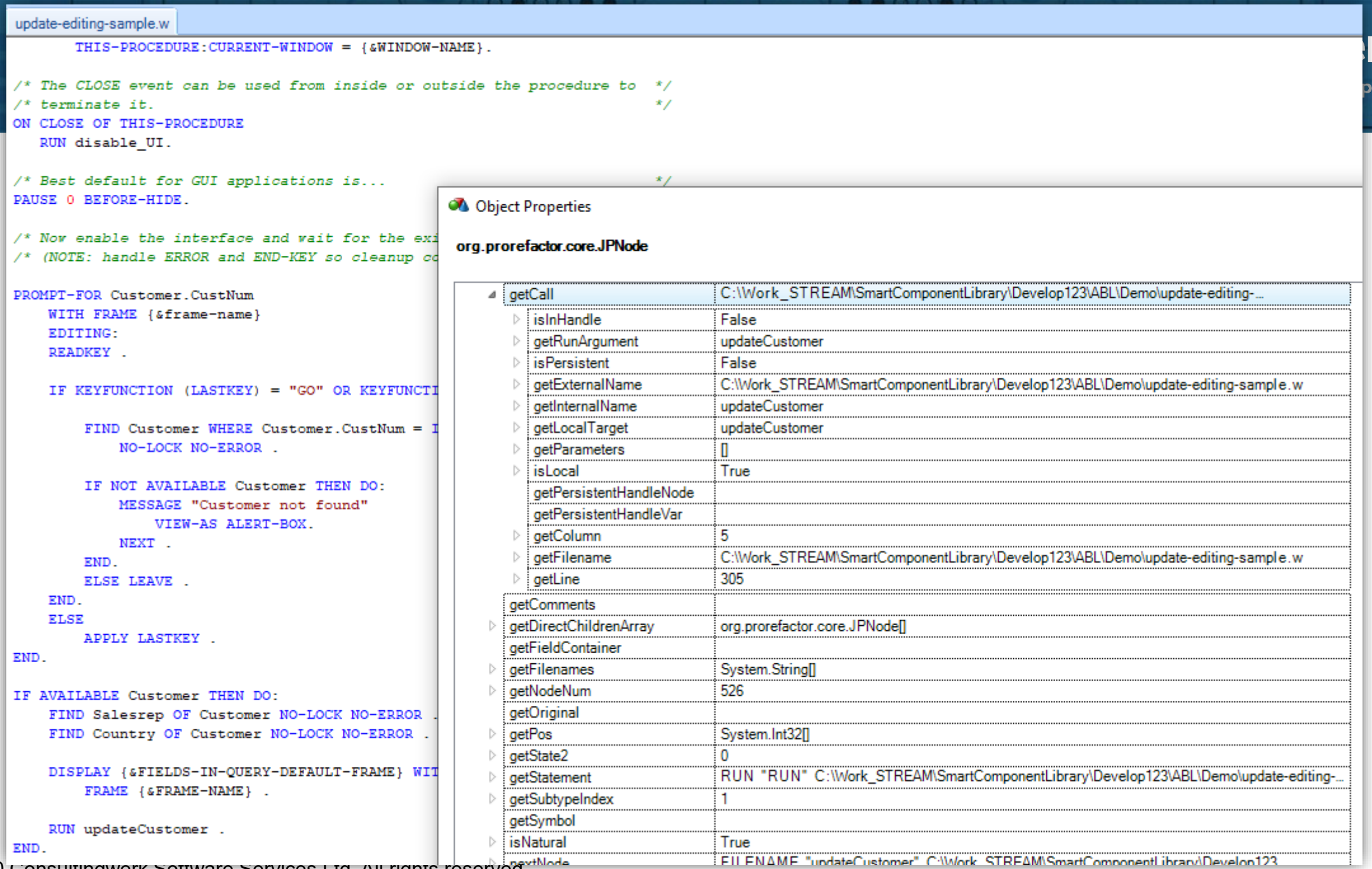
Constructors

Constructor and Description
`Call(JPNode node)`
`Call(java.lang.String externalName, java.lang.String internalName)` Construct a call to an internal procedure in a specific containing procedure.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	`addParameter(Parameter p)` Called by the tree parser.



Demo: Callgraph Utility

- Analyse und Darstellung von Call-Pfaden in ABL Applikationen

Bestehende Erweiterungen zum Proparse Call Objekt

- Aktuell in den ABL Implementierung unseres Callgraph Parsers implementiert, ggf. später Überführung zu Java-Code von Proparse
  - OOABL Referenzen zu Klassen und Methoden
  - Callgraph für Datenbanktrigger
  - SESSION Super-Procedure (simple Super-Procedures beherrscht Proparse out of the box)
- Handling von dynamischen RUN‘s
  - z.B. RUN VALUE über Callbacks um human-knowledge bereit zu stellen
  - ADM1 like RUN dispatch IN h_viewer („enable“) als Call zu local-enable, custom-enable, adm-enable oder enable

Geplante Erweiterungen

- Darstellen bestimmter Blöcke als Routinen, z.B. REPEAT, FOR EACH oder IF THEN DO: END.
- Sinnvoll falls eine Trigger-Block UI Handling und Datenbankzugriffe, Transaktionen mischt
- Grundlage für das (teil-)automatisierte Aufsplitten von Routinen in Frontend und Backend
- Unterstützt durch Proparse z.B. über die Scopes von Buffern und Symbolen (Variablen), welche dann in Parameter überführt werden müssen

Agenda

- Modernisierung von OpenEdge Anwendungen
- Quellcodeanalyse
- Proparse
- Use-Case 1: UI Layout überführen
- Use-Case 2: Validierungscode erkennen und migrieren
- Use-Case 3: Erkennen von Call-Pfaden
- **Schlussfolgerung**

Schlussfolgerung

- Proparse wird von uns seit Jahren als Grundlage für die Erstellung von Migrationsroutinen genutzt
- Migrationsroutinen profitieren von bestehenden Tools und API‘s
  - Zusätzliche Funktionalität mit fast jedem Migrationsprojekt
- Anpassbarkeit auf vorliegenden Code erforderlich und vorgesehen
- Routinen auf Basis von Proparse können viele Teilaspekte von Code-Refactoring beschleunigen
- Balance von Geschwindigkeit der Migration und Wartbarkeit des Ergebnisses
- 1-Click Migration einer 30 Jahre alten ABL Applikation ist Utopie

Haben wir Ihr Interesse geweckt? ☺

- Interesse heraus zu finden, wie Proparse auch Ihre Modernisierungsvorhaben beschleunigen kann?
- Gerne finden wir dies in einem unverbindlichen Gedankenaustausch mit Ihnen heraus!
- Sprechen Sie uns an! info@consultingwerk.de

Fragen