Erweiterungsentwicklung für Microsoft Dynamics 365 Business Central unter Verwendung von ExtensionsV2

Johannes Naderer
(se0307)

Inhaltsverzeichnis

Kapitel 1

Kurzfassung

Kapitel 2

Abstract

Kapitel 3

Einleitung

3.1 Motivation

ERP-Systeme sind heute aus dem wirtschaftlichen Umfeld nicht mehr wegzudenken [2] [1]. Bereits in den 1970er Jahren erkannten Wirtschaftstreibende Potential darin, ihre Prozesse und Unternehmensdaten zu digitalisieren. Während in den 1970er und 1980er Jahren innerhalb der einzelnen Abteilungen eines Unternehmens verschiedene Software-Lösungen zum Einsatz kamen, ist man sich mittlerweile einig, dass eine zentrale Applikation zur Verwaltung aller unternehmerisch wichtigen Daten und Prozessschritte große Vorteile liefert. Diese Erkenntnis hatte zur Folge, dass kleine Softwarehersteller immer mehr ins Wanken gerieten, da von der Wirtschaft allumfassende Software-Giganten gefordert wurden, die eine Vielzahl von Anforderungen aus den verschiedensten Anwendungsdomänen zu erfüllen haben. Solche Systeme können mit den meist begrenzten Ressourcen und Branchenwissen kleinerer Hersteller nicht entwickelt werden. Gleichzeitig entstanden durch die gewachsenen Anforderungen umfassende Softwaresysteme einiger größerer Hersteller, hier sind vor allem Marktführer SAP, aber auch Microsoft mit seiner Dynamics Sparte zu nennen.

Wer heute in einem Unternehmen mit der Einführung eines ERP-Systems betraut wird, muss sich intensiv mit den verschiedenen erhältlichen Lösungen auseinander setzen[2]. Denn neben Lizenzierung und finanziellen Aspekten, ist zu erarbeiten, welche Systeme die bestehenden Prozesse des Unternehmens am Besten abbilden. Systeme bilden Geschäftsprozess meist auf eine bestimmte Art ab. Sollte diese nicht mit dem Vorgehen des Unternehmens überein stimmen, bleiben meist nur zwei Auswegen offen. Entweder das Unternehmen passt seine Prozesse an die Vorgabe des Systems an, oder das System muss entsprechend angepasst werden, um den Ansprüchen des Unternehmensprozesses zu genügen.

Und genau hier spielt die Anpassbarkeit und Erweiterbarkeit eines Systems die zentrale Rolle. Gerade branchenspezifische und insbesondere unternehmensspezifische Prozesse müssen meist erst programmiert und in das System integriert werden. Programmierarbeiten und Änderungen am Standardsystem sind meist Aufwendig, und stellen so ein nicht zu vernachlässigendes finanzielles Risiko dar.

Um die Aspekte der Erweiterbarkeit und Anpassungsmöglichkeit möglichst gut zu erfüllen, entschied sich Microsoft im ERP-System Microsoft Dynamics NAV bereits in

sehr frühen Versionen, zertifizierten Entwicklern freien Zugang zum Applikationscode zu gewähren[3]. So können Entwickler die gesamte Geschäftslogik des Systems je nach Unternehmensanforderungen abändern und erweitern, in dem Sie neuen Programmcode hinzufügen, oder den Standardcode von Microsoft anpassen oder löschen. Dies hat zum Einen zur Folge, dass Entwickler mächtige Applikationen erstellen können, und sich diese direkt in das bestehende System integrieren lassen. Allerdings kommen mit diesen umfassenden Möglichkeiten auch Probleme auf. Je weiter die oft über Jahrzehnte verwendeten und erweiterten Systeme von der Codebasis von Microsoft abweichen, desto aufwendiger, fehleranfälliger und teurer ist es, diese Systeme mit den Aktualisierungen des Herstellers zu versorgen, die periodisch in das System integriert werden müssen.

Um die Systeme update-fähig zu halten, und gleichzeitig ein Entwicklungsmodell zu schaffen, dass auch in der Cloud-Variante des ERP-Systems funktionieren kann, wurde mit Dynamics NAV 2017 erstmals das Konzept der Erweiterungsprogrammierung mit ExtensionsV1 für Dynamics NAV vorgestellt. ExtensionsV1 ist ein gänzlich neuer Ansatz für das ERP-System zu programmieren, und hat konzeptionell viele Vorteile gegenüber der konventionellen prozeduralen Entwicklung, ist aber mittlerweile aufgrund einiger technischer Schwierigkeiten obsolet.

Mit der in 2018 veröffentlichten Version - ExtensionsV2 - sind nicht nur viele der technischen Mängel behoben, ExtensionsV2 kommt auch mit einer neuen Programmiersprache und Entwicklungsumgebung.

3.2 Zielsetzung

Im Rahmen dieser Arbeit, wird ein Überblick über die Plattform Dynamics NAV bzw. die Cloud-Variante Dynamics 365 Business Central und die Programmierung dieser Systeme gegeben. Hierfür wird erst eine Übersicht über das Gesamtsystem und seine Schichtenarchitektur vermittelt. Anschließend wird das Konzept der Erweiterungsentwicklung mit ExtensionsV2 mit der konventionellen prozeduralen Entwicklung verglichen. Dies erfolgt anhand eines Beispiels, dass auf beide Arten gelöst wird. Einerseits wird die Anwendung in der Entwicklungsumgebung C/SIDE mit C/AL entwickelt. Andererseits wird anhand der Aufgabenstellung eine Erweiterung mit ExtensionV2 in VisualStudio Code und der aktuellen Sprache AL erstellt. Hierbei liegt der Fokus nicht darauf, kleine syntaktischen Unterschiede zwischen den Sprachen hervorzuheben, sondern Neuerungen in der Sprache AL zu beleuchten, und konzeptionelle Unterschiede zwischen den beiden Programmierparadigmen aufzuzeigen und zu bewerten.

Der Vergleich erfolgt anhand von statischen und dynamischen Code-Metriken sowie Laufzeitmessungen. Zusätzlich wird auch diskutiert, welche Vor- und Nachteile sich durch die nun neue Datei-basierte Codeverwaltung hinsichtlich der Einbindung und Nutzung von Source Code Management und Continuous Integration Systemen ergeben. In einem letzten Block wird danach das Event-basierte Programmiermodell der Erweiterungsentwicklung mit ExtensionsV2 diskutiert, die Vor- und Nachteile beleuchtet, die mit dem Wechsel von Code-Anpassung hin zu Code-Erweiterung einher gehen.

Kapitel 4

Stand der Technik

4.1 ERP: Definition und Übersicht

ERP Systeme sind umfangreiche kommerzielle Softwaresysteme, mit der Kernaufgabe, alle Abteilungen und Prozesse eines Unternehmens soweit wie möglich digital abzubilden[6]. Ein ERP-System stellt für ein Unternehmen somit eine zentrale Verarbeitungs- und Datensicherungsplattform für sämtliche unternehmensrelevanten Geschäftsdaten bereit. Die Daten sind hierbei in einem einzelnem System erfasst, und sind so sofort für alle Unternehmensbereiche verfügbar. Es sind keine Synchronisierungsschritte oder Schnittstellen innerhalb des Systems nötig. Durch die zentrale Datenerfassung, stellt ein ERP-System eine durchgängige Informationsquelle für alle Unternehmensbereiche dar, die bei der Prozessanalyse und Analyse als Datenbasis für geschäftliche Entscheidungsträger unabdingbar ist.

Im Gegensatz zu abteilungsbezogenen Systemstrukturen (Insellösungen) ist es durch Einsatz eines ERP-Systems möglich Funktionen zu nutzen, für deren Durchführung Informationen aus mehreren Abteilungen nötig sind[5]. So kann zum Beispiel bei Eingang eines Auftrags sofort automatisiert geprüft werden, ob der Auftrag angenommen werden soll. Entscheidungen wie diese basieren auf einem sehr breiten Datenstamm aus den verschiedenen Abteilungen. Aus den Daten der Finanzabteilung können Zahlungsmoral, offene Beträge des Kunden und ein voraussichtlicher Deckungsbeitrag eine Rolle für diese Entscheidung spielen. Anhand der Lagerhaltungsdaten kann sofort eine Verfügbarkeitsprüfung für die bestellten Artikel durchgeführt werden. Mithilfe von Produktions- und Personaldaten wird ausgewertet, ob ausreichend Personal und Maschinenressourcen für die Erfüllung des Auftrags zur Verfügung stehen. Dies sind nur einige wenige Beispiele, wie ein ERP-System bei der täglichen unternehmerischen Tätigkeit behilflich sein kann.

Da in ERP-Systemen der Zugriff auf die Datenbank nicht durch die Systemarchitektur eingeschränkt ist, muss der Zugang zu den Daten im System über ein Rechte- und Modulsystem gesteuert werden[5]. Berechtigungssätze lassen sich hier meist sehr feingranular definieren, sodass einerseits der Schutz sensibler Daten gewährleistet ist, jedoch andererseits alle benötigten Daten entsprechend betrachtet und verarbeitet werden können.

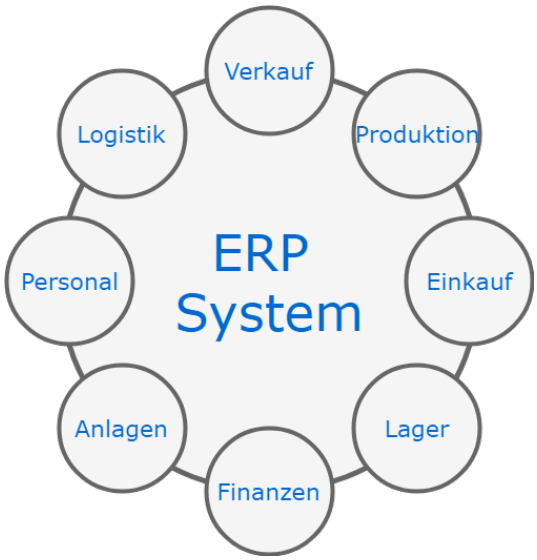

Abbildung 4.1: Schematische Darstellung: Modularisierung anhand Unternehmensabteilung

Um die umfangreichen Funktionalitäten eines ERP-Systems zu gliedern und aufzuteilen, bedienen sich die meisten Hersteller eines Modul-Systems. Meist spiegelt die Aufteilung dieser Module die einzelnen Abteilungen eines Unternehmens wieder. So verteilt sich die Gesamtfunktionalität eines Systems beispielsweise auf ein Einkaufsmodul, ein Vertriebsmodul und viele andere Teilbereichsmodule auf. Diese Module können in ihren Grundzügen unabhängig voneinander verwendet werden. So kann ein Unternehmen beispielsweise entscheiden, vorerst nur Finanzen und Personal über das System zu verwalten. Andere Module können im Laufe der Zeit stückweise in Betrieb genommen werden. Zu den bekanntesten ERP Herstellern zählen unter anderen IBM, SAP, Microsoft, Infor und Sage.

4.2 Geschichte und Grundarchitektur Dynamics 365 Business Central

4.2.1 Geschichte

Das ERP-System, dass heute *Microsoft Dynamics 365 Business Central* heißt, erschien ursprünglich 1984 unter dem Namen *PCPlus* als ein ERP System für Microsoft DOS in Dänemark[4]. Während seiner mittlerweile 35-jährigen Geschichte wurde das Produkt einige Male neu benannt und an den technischen Fortschritt angepasst. Was 1984 begann, wurde 1995 unter dem Namen *Navision Financials* als das erste ERP-Produkt mit grafischer Benutzeroberfläche für Windows95 präsentiert. 2002 wurde *Navision Financials* von Microsoft gekauft und unter dem Namen *Microsoft Business Solutions Navision* vertrieben. In all diesen Jahren basierte die Datenspeicherung des Systems in einem komplexen Dateibasierten Format. Im Jahr 2008 passiert dann der Schritt zu Microsoft SQL Server und der 3-Schichten-Architektur, nun unter dem Namen *Microsoft Dynamics NAV 2009*. Der vorerst letzte Meilenstein in der Geschichte des Systems ist 2018. Das System wird nun als Cloud-ERP-System unter dem Namen *Microsoft Dynamics 365 Business Central* betrieben.

Trotz den vielen Versionen und der jahrzehntelangen Geschichte dieses Systems, finden sich auch in der heutigen Code-Basis noch viele Passagen, die bereits in den 1980er Jahren entstanden, und bis heute produktiv eingesetzt werden.

4.2.2 3-Schichten Architektur

Dynamics 365 Business Central basiert auf einer 3-Schichten Architektur. Durch Schichtenarchitekturen lassen sich die Aufgabengebiete bzw. Teile eines komplexen Softwaresystems aufteilen. Im Falle von Dynamics Business Central 365 unterscheiden wir zwischen der Endbenutzerschicht, Serverschicht und Datenbankschicht.

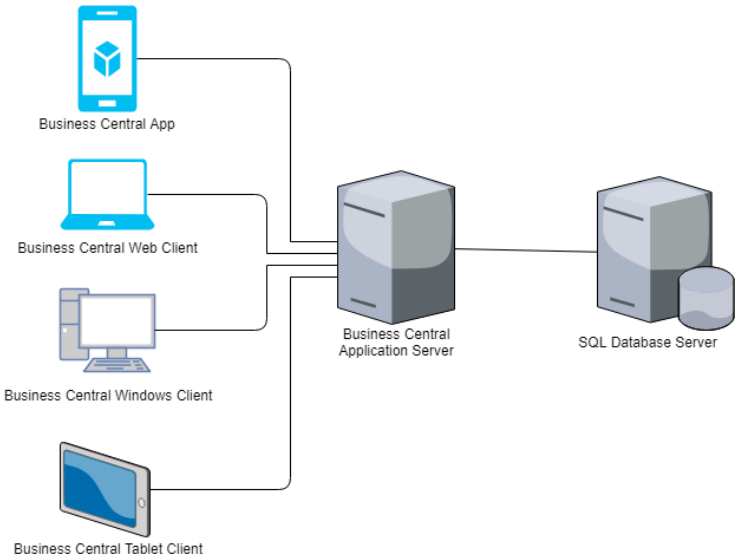

Abbildung 4.2: Dynamics 365 Business Central: 3-Schichten Architektur

Schicht 1: Die Endbenutzerschicht/Präsentationsschicht: In dieser Schicht der Architektur finden sich sämtliche Softwarekomponenten, die direkt die von der Serverschicht exportierten Funktionalitäten nutzen. Hierzu zählen vorrangig die von Microsoft veröffentlichten Endbenutzerprogramme wie der Business Central WebClient und die Business Central Mobile App. Aber auch von Drittanbietern erstellte Softwarekomponenten, die Microsoft Graph API oder von Business Central veröffentlichte Webdienste nutzen sind Teil der Endbenutzerschicht.

Schicht 2: Die Serverschicht: Die Business Central Serverapplikation (auch *Middle-Tier* oder *Service-Tier*) stellt das Herzstück des Gesamtsystems dar. Der Business Central Server ist eine .NET basierte Serveranwendung und nutzt die Windows Communication Foundation (WCF) als Kommunikationsprotokoll. Der Server nimmt sämtliche Anfragen von Endbenutzerprogrammen entgegen, holt anhand dieser Anfragen Daten von der Datenbankschicht ab, führt mithilfe der abgeholten Daten Geschäftslogik aus, bereitet die Ergebnisse der Geschäftslogik auf, und liefert diese zurück an das anfragende Endbenutzerprogramm. Neben den Endpunkten zur Client-Kommunikation beinhaltet die Serverkomponente auch die Webserverkomponenten zur Nutzung des WebClients. Die Webserverkomponente selbst ist eine ASP.NET Core Applikation, die auf einem mitgeliefertem IIS (Internet Information Server) läuft. Daher ergibt sich auch die Anforderung, dass die Serverschicht auf einer Windows-Server Maschine betrieben werden muss.

Schicht 3: Die Datenbankschicht: Hinter der Datenbankschicht verbirgt sich eine Microsoft SQL Server Instanz. Hierbei ist zu erwähnen, dass aufgrund der historischen Entwicklung des Gesamtsystems einige Funktionen einer klassischen relationalen Datenbank hier nicht verwendet werden. So wird man am Datenbankserver vergeblich nach Relationen zwischen Tabellen suchen (Fremdschlüsselbeziehung), denn diese existieren hier schlichtweg nicht. Diese Beziehungen werden von der Serverschicht verwaltet. Auf Grund dieser Tatsache ist es strengstens abzuraten manuell mit SQL Befehlen Datenbestände zu ändern. Änderungen, die nicht durch die Logik der Serverschicht validiert werden können, können schnell zu Inkonsistenzen in den Daten führen, die in weiterer Folge das Gesamtsystem korrumpieren und zu Systemausfällen führen können.

Die Unterteilung der einzelnen Teilbereiche des Systems in drei Teilbereiche liefert einige Vorteile. So können anhand der vom Server exportierten Schnittstellen schnell neue Apps und

4.3 Objektarten in Business Central

Die Programmierung von Microsoft Dynamics 365 Business Central erfolgt durch die Erstellung und Anpassung von Applikationsbauteilen die gemeinhin *Objekte* genannt werden. Diese *Objekte* sind nicht mit jenen aus der klassischen objektorientierten Programmierung zu vergleichen. Dynamics 365 ist objektbasiert und nicht objektorientiert. Entwickler haben nicht die Möglichkeit neue Typen von Objekten zu erstellen, sondern nur neue Ausprägungen der bestehenden Objekttypen zu entwickeln. AL bietet gegenüber C/AL neue Möglichkeiten zur Entwicklung unter AL, was sich auch in den verfügbaren Objekttypen ausdrückt.

Type	C/AL	AL
Table	X	X
TableExtension		X
Page	X	X
PageExtension		X
Report	X	X
Codeunit	X	X
Query	X	X
XMLPort	X	X
MenuSuite	X	
Enum		X

Tabelle 4.1: Verfügbarkeit Objekttypen: C/AL und AL

Table: Tabellenobjekte definieren den Datenaufbau, Restriktionen und Validierungsregeln für alle Daten in Microsoft Dynamics 365 Business Central. Beim Kompilieren von Tabellenobjekten, veranlasst die Serverschicht die Erstellung oder Änderung einer zum Tabellenobjekt gehörigen SQL-Tabelle, in der schlussendlich die Daten landen. Tabellenobjekte sind aber weit mehr als lediglich eine Beschreibung zur Datenhaltung. Tabellenobjekte definieren auch sämtliche Validierungslogik, sowohl auf Feld- als auch auf Datensatzebene. Darüber hinaus gibt es auch die Möglichkeit, innerhalb von Tabellenobjekten Funktionen und Prozeduren für die beschriebenen Daten zu definieren.

TableExtension: Tabellenerweiterungen sind ein zentraler Baustein der Erweiterungsentwicklung mit AL. Sollten an einer Tabelle Änderungen oder Erweiterungen nötig sein, würde man mithilfe C/AL einfach das bestehende Tabellenobjekt abändern. Mit dem Konzept der Erweiterunsentwicklung und AL ist dies nicht mehr möglich. Genau hier kommen Tabellenerweiterungsobjekte ins Spiel. Mithilfe von Tabellenerweiterungen lassen sich Felder und Logiken eines Tabellenobjektes erweitern, ohne das Tabellenobjekt selbst zu ändern. Beim Kompilieren von Tabellenerweiterungen wird auf SQL-Seite zusätzlich zur Basistabelle eine zusätzliche hinzugefügt (textitCompanion Table). Diese *Companion Table* verfügt über den selben Primärschlüssel wie die Basistabelle, und bietet Platz für Felder der Erweiterung. Auf SQL-Seite wird für die Benennung der *Companion Table* der Name der Basistabelle um die eindeutige GUID der Erweiterung

ergänzt. Tabellenerweiterungen sind nur unter AL verfügbar.

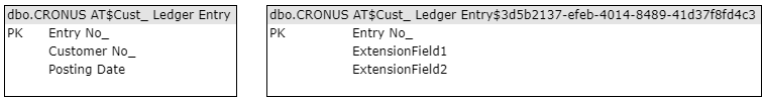

Abbildung 4.3: Dynamics 365 Business Central: Standardtabelle und dazugehörige Tabellenerweiterung

Pages: Seiten sind unter Dynamics 365 Business Central der Baustein zur Erstellung grafischer Benutzeroberflächen. Im Vergleich zu anderen Systemen hat man hier jedoch keinen bedeutenden kreativen Freiraum bei der Gestaltung. Seiten definieren lediglich die angezeigte Information, deren Gruppierung, Sortierung und die anwendbaren Aktionen auf diese. Die visuelle Gestaltung wird bis auf wenige Ausnahmen vom System vorgegeben, sodass man als Entwickler in diesem Aspekt nur sehr eingeschränkte Möglichkeiten hat. Diese Art der Einschränkung ist zwar auf den ersten Blick als Nachteil zu betrachten, garantiert dem Nutzer des Systems jedoch ein durchgängiges Design der Benutzeroberflächen.

PageExtension: Mithilfe von Seitenerweiterungen lassen sich bestehende Seiten anpassen und ergänzen, ohne das ursprüngliche Seitenobjekt abzuändern. Der Benutzer merkt bei Betrachtung des Ergebnisses nicht, dass es sich hierbei um ein zusätzliches Objekt handelt. Rein optisch integrieren sich Seitenerweiterungen nahtlos in ihre Ursprungsobjekte.

Report: Berichte erlauben es, Auswertungen und Dokumente aus dem System zu generieren, etwa eine Bilanzübersicht, oder eine Verkaufsrechnung. Dabei bestehen Berichte aus zwei Komponenten: einem Dataset und einem Layout. Im Dataset werden die im Layout zur Verfügung stehenden Daten definiert. Das Layout selbst bestimmt die grafische Repräsentation dieser. Durch die Integration von Datasets in Microsoft Word Vorlagen, lassen sich durch den Nutzer auf einfache Weise benutzerdefinierte Word-Layouts erstellen.

Codeunit: Codeunits stellen die Logikkomponenten des Systems dar. Sämtliche Berechnungen, Buchungsroutinen und andere Teile der Geschätslogik finden sich hier. Um die Funktionalität einer Geschäftslogik abzuändern, ist es unter C/AL üblich direkt den verantwortlichen Code dafür abzuändern. Unter AL können bestehende Codeunits nicht geändert werden. Um unter AL Änderungen durchzuführen, müssen Event-Subscriber erstellt werden, die sich in die Standardlogik einklinken.

Query: Abfrageobjekte bieten die Möglichkeit, hochperformante Abfragen an die Datenbank abzusetzen. Die Ergebnisse dieser Abfragen werden meist als Dataset für Berichte genutzt, oder als WebService exportiert.

XMLPort: XMLPorts bieten eine einfache Möglichkeit, Daten in das System zu importieren und aus dem System zu exportieren. Trotz des Objektnamens, sind XMLPorts nicht nur auf XML beschränkt, sondern können auch verwendet werden um CSV und andere Formate zu verarbeiten.

MenuSuite: MenuSuite-Objekte bestimmen die hirarchische Menüführung des Systems, und bestimmen wie andere Objekte über die grafische Oberfläche aufgerufen werden können. MenuSuites sind nur unter C/AL verfügbar, unter AL werden die nötigen Informationen dafür direkt in den Seiten- und Reportobjekten selbst definiert.

Enum: Nur unter AL verfügbar. Enums definieren Aufzählungstypen und ersetzen den in C/AL verwendeten *Option* Datentyp, der lediglich eine kommaseperierte Zeichenfolge darstellt, schlecht erweiterbar ist, und somit für die Erweiterungsentwicklung unter AL nicht zielführend ist.

Kapitel 5

Vergleich

Folgend wird zur Darstellung des Entwicklungsprozesses, und der Unterschiede zwischen der konventionellen prozeduralen Entwicklung in C/AL, und der erweiterungsbasierten Entwicklung in AL ein Beispiel in beiden Sprachen entwickelt.

5.1 Aufgabenstellung

Für die Kunden des Auftraggebers unserer Erweiterung sollen Treuepunkte verwaltet werden. Treuepunkte werden mit dem Kauf von Waren verdient, oder von der Marketingabteilung an Bestandskunden vergeben. Treuepunkte können beim Kauf von Produkten eingelöst werden, um einen Preisnachlass zu erzielen. Eingelöste Punkte verringern den Rechnungsbetrag um einen bestimmten Geldwert, der variieren kann. So mag ein Treuepunkt im Januar 10 Cent wert sein, im Februar jedoch 15 Cent. Die Schwankung des Treuepunktwertes wird als Marketinginstrument genutzt. Auch wie viele Treuepunkte beim Einkauf vergeben werden ist variabel, so sind etwa Aktionszeiträume vorgesehen, in denen beim Einkauf doppelt so viele Treuepunkte verdient werden können.

Das neue Treuepunktesystem ist für das Marketing von hoher Bedeutung. So ist es erforderlich, dass Änderungen am Treuepunktekonto eines Kunden über einen Web Service an das verwendete CRM-System gemeldet werden. Für das Reporting im Unternehmen ist es außerdem nötig, dass täglich ein XML Datenträger erzeugt werden kann, in dem der Treuepunktesaldo und die Bewegungen des aktuellen Tages je Kunde ersichtlich sind. Zusätzlich zu dieser Datei für das Berichtssystem soll auch ein übersichtlicher Ausdruck in PDF-Form an die Marketingleitung gesendet werden.

5.2 Entwicklungsprozess

5.2.1 Entwicklungsumgebung

C/AL - Development Environment

C/AL wird im *Microsoft Dynamics Development Environment* entwickelt. Dabei handelt es sich eigentlich um den Client, der bis zur Version 2009 noch als Windows Client für Endbenutzer verwendet wurde, nun seit dem jedoch rein für die Entwicklung genutzt wird. Das Development Environment ist ein Windows Client, der stets sowohl mit Datenbank, als auch mit der Serverapplikation verbunden sein muss. Die Datenbankverbindung ist nötig, da darin die Applikationsobjekte gespeichert sind, die Verbindung zum Applikationsserver, um Änderungen kompilieren und ausführen zu können.

Das Kernstück des Development Environment bildet der *Object Designer*. Der *Object Designer* liefert einen Überblick über sämtliche, im System vorhanden Applikationsobjekte und Details zu Ihnen. Ein Applikationsobjekt unter C/AL wird durch eine numerische ID und seinen Namen identifiziert. Zusätzlich wird zu den einzelnen Applikationsobjekten auch gespeichert, ob und wann sie das letzte Mal geändert wurden.

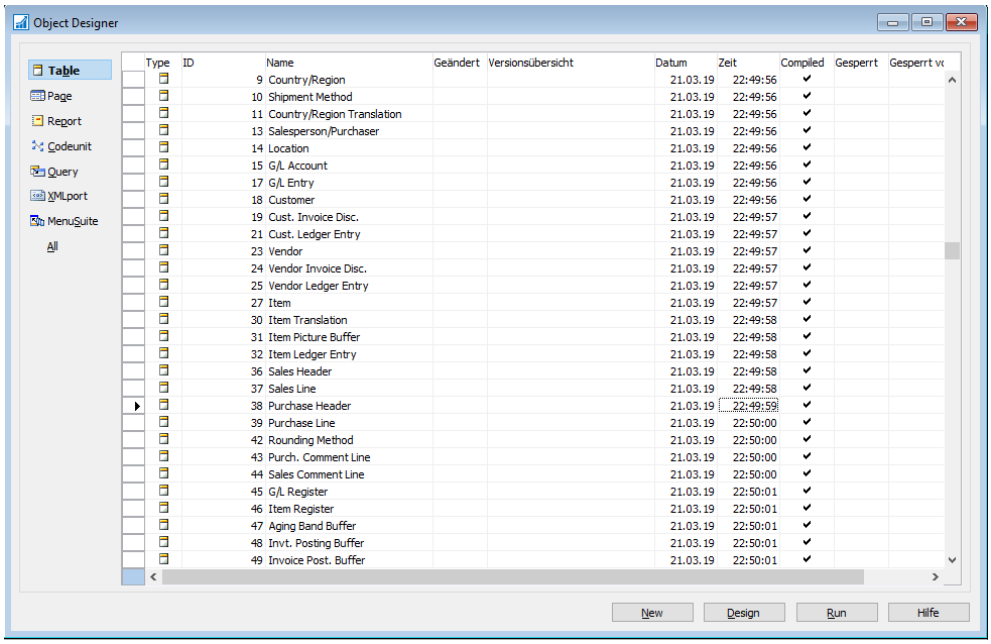

Abbildung 5.1: Development Environment: Object Designer und C/AL Editor

Je nach ausgewählter Objektart stellt das Development Environment einen auf die Objektart angepassten *Designer* zur Verfügung, über den bereits einige Basiseinstellungen getätigt werden können. Im Falle von Tabellenobjekten, können mithilfe des Table Designers Tabellenfelder angelegt, entfernt und bearbeitet werden. Über den Designer

gelangt man ebenfalls zum C/AL Editor, in dem die Implementierung der Geschäftslogik passiert.

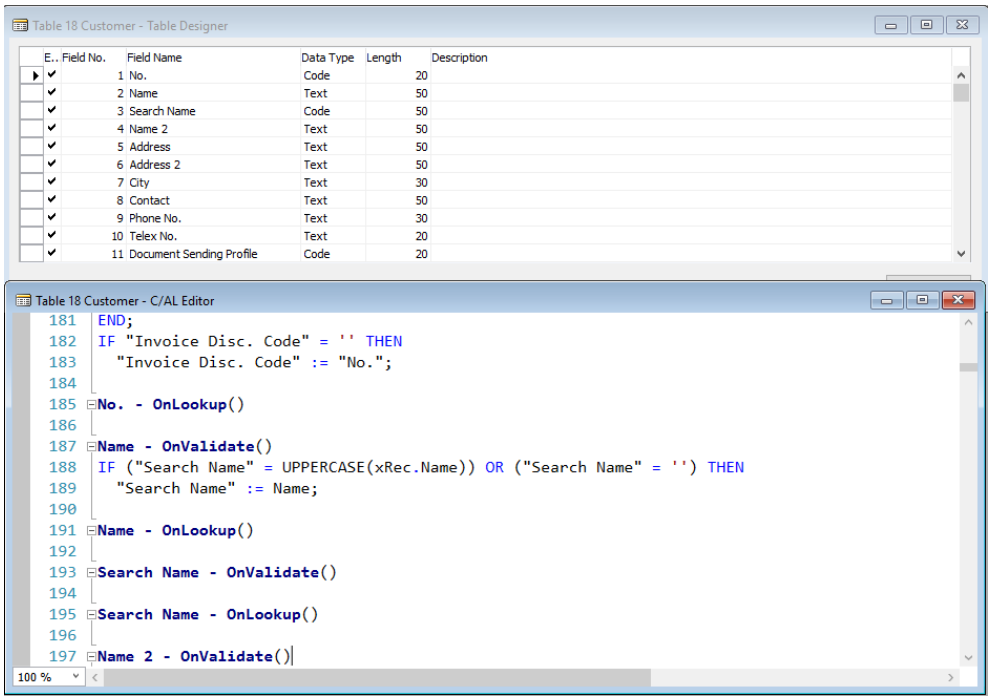

Abbildung 5.2: Development Environment: Table Designer und C/AL Code Editor der Debitoren Tabelle

Die Sprache C/AL basiert auf Pascal. Im Gegensatz zu Pascal ist C/AL jedoch rein prozedural, und rein auf die Arbeit mit Dynamics NAV bzw. Business Central spezialisiert. So bietet C/AL mithilfe der inkludierten *Record API* eine einfache und effiziente Weise, Datensätze aus der Datenbank zu lesen, filtern, schreiben und zu löschen. Der Mehraufwand der in anderen Sprachen und Systemen durch die Erstellung von Datenbankverbindungen verursacht wird ist in C/AL minimal, da die Datenbankverbindung bereits durch die Verbindung zum Server gegeben ist. Der Datenbankkontext kann daher vom Applikationsserver bestimmt werden, und muss nicht im Applikationscode definiert werden. Andererseits fehlen innerhalb C/AL Funktionalitäten, die in modernen Programmiersprachen mittlerweile zum Standardumfang fehlen, wie zum Beispiel eine Möglichkeit zur Kommunikation via HTTP.

Um Applikationsobjekte zwischen verschiedenen Datenbanken zu transferieren, um beispielsweise entwickelte Applikationsobjekte von einem Testsystem in die Produktivumgebung zu übernehmen, bietet das Development Environment die Möglichkeit Applikationsobjekte zu exportieren. Dieser Export kann in zwei verschiedenen Formaten erfolgen. Einerseits im Textformat. Dabei werden sowohl Code als auch die im Designerfenster getätigten Einstellungen in ein spezielles Textformat gebracht. Dieses Textformat (Dateiendung .txt) ist zwar grundsätzlich durch den Menschen lesbar, definiert jedoch

auch einige intern nötige Eigenschaften. Andererseits können Objekte auch im Binärformat exportiert werden (Dateiendung .fob). Das Binärformat zeichnet sich im Gegensatz zum Textformat durch geringere Dateigröße und besserer Performanz beim Importieren und Exportieren aus, und ist daher das Standardformat um Applikationsobjekte zwischen Datenbanken zu transferieren.

AL - Visual Studio Code

Die Programmierung von Erweiterungen für Microsoft Dynamics 365 Business Central erfolgt in Visual Studio Code [7]. Visual Studio Code ist ein OpenSource Quelltext-Editor für verschiedenste Programmier- und Markupsprachen basierend auf dem Electron Framework. Visual Studio Code ist in der Sprache Typescript implementiert. Im Gegensatz zum Development Environment setzt Visual Studio Code kein Windows-Betriebssystem voraus, sondern kann auch unter Mac und Linux verwendet werden. Als zeitgemäße Entwicklungsumgebung liefert Visual Studio Code eine Auswahl einiger Features für Entwickler, die im Development Environment nicht vorzufinden sind. Darunter:

- Refactoring Werkzeuge
- IntelliSense und Code-Vervollständigung
- Mauslose Bedienung
- Integrierte Source Code Verwaltung mit Git
- Erstellung von benutzerdefinierten Tastenkombinationen

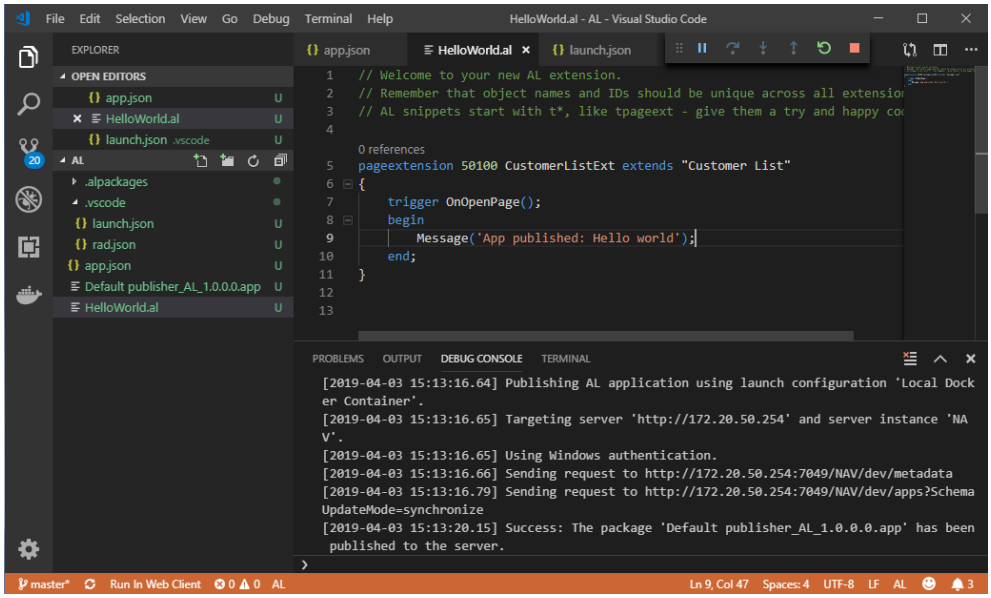

Abbildung 5.3: Visual Studio Code: Grafische Oberfläche, Hello World Extension

Im Gegensatz zum Development Environment ist Visual Studio Code nicht dafür ausgelegt, mit Business Central und AL zu Arbeiten. Eine Basisinstallation von Visual Studio Code kann so auch nicht für die Entwicklung unter AL genutzt werden. Hier kommt jedoch eine große Stärke von Visual Studio Code ins Spiel, seine Erweiterbarkeit. Mit Business Central wird die dazugehörige Visual Studio Code Erweiterung mitgeliefert, die für die Entwicklung von AL-Erweiterungen nötig ist. Diese Erweiterung *AL Language Extension*, wird im .vsix Format von Business Central zur Verfügung gestellt und lässt sich mittels weniger Klicks installieren.

Visual Studio Code wird monatlich automatisch mit Updates versorgt. Auch Neuheiten für die AL Spracherweiterung werden automatisch mitinstalliert, wobei es einfach möglich ist, frühere Versionen der Erweiterung zu verwenden um auch mit Systemen arbeiten zu können, die noch nicht auf dem neuesten Stand sind. Dies stellt für Entwickler einen bedeutenden Vorteil dar, da das Development Environment bei Neuerungen immer manuell geladen werden musste, und mehrere lokale Installationen nötig waren, um auch vorangegangene Versionen des Systems zu unterstützen.

Visual Studio Code in Kombination mit AL ist rein textbasiert. Die aus dem Development Environment bekannten verschiedenen Designer Fenster finden unter AL keine Anwendung mehr. Applikationsobjekte werden nicht mehr direkt aus der Datenbank gelesen und zurückgeschrieben, sondern existieren nun zur Entwicklungszeit als Dateien in einem Verzeichnis auf der Entwicklermaschine. Somit sind keine proprietären Exportmechanismen mehr nötig, die Datei beinhaltet sämtliche Informationen für das spätere Laufzeitobjekt, und wird als solches komplett vom Entwickler verfasst. Im Gegensatz zum Textexport aus dem Development Environment steht in den erstellten AL Dateien genau was der Entwickler vorgibt. Nicht mehr und nicht weniger. Dies ist einer der größten Vorteile, die die neue Entwicklungsumgebung mit sich bringt. Denn dadurch lässt sich der geschriebene Quellcode sinnvoll und ohne Umwege in einem Source Code Management System wie Git verwalten.

5.2.2 Grundfunktionalität

Um die Grundfunktionalität der Treuepunkterweiterung, das Sammeln und Einlösen von Treuepunkten abzubilden, sind einige Änderungen und Ergänzungen an der Standard-Tabellenstruktur von Dynamics 365 Business Central nötig.

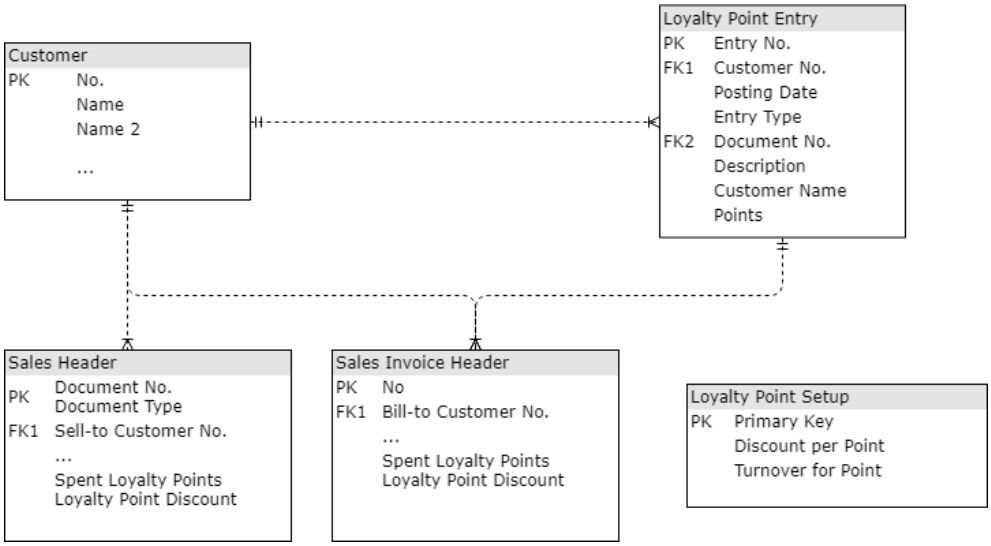

Abbildung 5.4: Grundfunktionalität: Tabellen

Es wird eine neue Tabelle *Loyalty Point Entry* eingeführt, in der alle Transaktionen betreffend Treuepunkten gespeichert werden. Um diverse Auswertungen zu ermöglichen, werden neben der betroffenen Punktezahl, und dem zugehörigen Kunden auch die Dokumentennummern und die Transaktionsart (*Entry Type*) gespeichert. Die Transaktionsart kann dabei einen von drei Werten annehmen: Verdienst, Einlösung und Marketing. Um die Treuepunkterweiterung konfigurierbar zu machen wird zusätzlich auch eine Setup-Tabelle *Loyalty Point Entry* erstellt. Diese folgt dem in Dynamics 365 Business Central oft gebrauchten Softwaremuster der Setup-Tabelle. Dabei handelt es sich um eine Tabelle mit einem Primärschlüssel *Primary Key*, in der maximal ein Datensatz gespeichert werden kann. In Tabellen dieser Art werden Konfigurationen getroffen, um andere Bereiche der Geschäftslogik zu parametrisieren.

- *Discount per Point*: Bestimmt für wieviele Währungseinheiten ein Treuepunkt eingelöst werden kann.
- *Turnover for Point*: Definiert, wieviel Nettoumsatz zur Vergabe eines Treuepunktes führt.

Bei Eingabe einer neuen Verkaufsrechnung an Kunden müssen die konfigurierten Parameter abgefragt werden, um entsprechend Rechnungsrabatte zu erteilen. Wird die Rechnung danach gebucht, muss zur Behandlung der Treuepunkte in die Buchungslogik eingegriffen werden. Einerseits muss vor dem Buchen geprüft werden, ob der Kunde auch genug verfügbare Treuepunkte hat, um den Rechnungsrabatt mit seinem Punkte-

konto ausgleichen zu können. Diese zusätzliche Prüfung ist wichtig, da Erfassung und Verbuchung der Rechnung nicht zwangsweise zum selben Zeitpunkt erfolgen müssen. Andererseits muss nach erfolgreichem Verbuchen der Rechnung das Treuepunktekonto des Kunden angepasst werden, dabei müssen sowohl eingelöste, als auch durch die Rechnung verdiente Punkte berücksichtigt werden. Vorbereitete (*ungebuchte*) Rechnungen sind Datensätze der Tabelle *Sales Header*, und werden von der Buchungsroutine zu Datensätzen der Tabelle *Sales Invoice Header* konvertiert. In beiden Tabellen werden die nötigen Felder zur Eingabe und Speicherung der notwendigen Daten für die Rabattvergabe hinzugefügt.

Für die Implementierung der Geschäftslogik wird ein entsprechendes Codeunit Applikatonsobjekt erstellt, deren Prozeduren aus allen anderen Applikationsobjekten aufgerufen werden können.

C/AL

Wir gehen von einer bestehenden Business Central Instanz, inklusive bestehender Server und Datenbankverbindung aus. Die nötigen beschriebenen Tabellenänderungen und neuen Tabellen können unter C/AL im Development Environment fast ausschließlich mithilfe des *Table-Designer* Fensters erstellt werden.

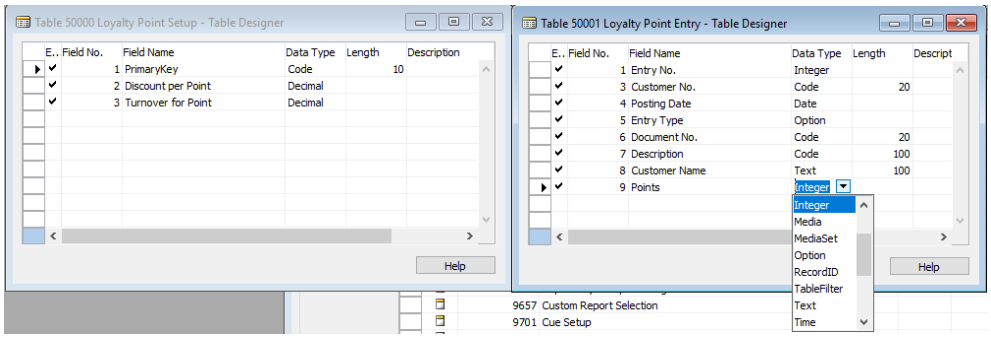

Abbildung 5.5: Grundfunktionalität: Tabellen der Treuepunkterweiterung

Beim Speichern der erstellten Tabellenobjekte, werden diese vom Server direkt kompiliert, und anschließend am Datenbankserver das Datenbankschema entsprechend erweitert. Neben den beiden in 5.5 erstellten Tabellen, sind auch Änderungen der Tabelle *Sales Header* und *Sales Invoice Header* nötig (siehe 5.4). Das Feld *Spent Loyalty Points* der Tabelle Sales Header ist für die Einlösung von Treuepunkten notwendig. Bei der Validierung dieses Feldes muss darauf geachtet werden, dass nicht mehr als die vorhandene Treuepunkte eingelöst werden können, und dass der Rechnungsrabatt der derzeitigen Rechnung korrekt anhand der Einrichtungstabelle gesetzt wird. Dies passiert mithilfe des folgenden C/AL Codes:

Programm 5.1: Validierung Treuepunkeinlösung

```
1 Trigger "Spent Loyalty Points" - OnValidate()
2
3 // Make the user can not assign more points, than the customer currently has
4 Customer.GET("Bill-to Customer No.");
5 Customer.CALCFIELDS("Loyalty Points");
6 IF "Spent Loyalty Points" > Customer."Loyalty Points" THEN
7   ERROR(ErrTooManyPointsAssigned, FORMAT(Customer."Loyalty Points"));
8
9 // Adjust Invoice discount in case there was a loyalty discount beforehand
10 IF "Loyalty Point Discount" <> 0 THEN
11   VALIDATE("Invoice Discount Amount",
12            "Invoice Discount Amount" - "Loyalty Point Discount");
13
14 // Calc new Discount, and assign it to the current invoice
15 // Do not assign the discount directly , in case there are already other invoice discounts in
       place
16 VALIDATE("Loyalty Point Discount",
17          LoyaltyPointMgmt.CalculateDiscount("Spent Loyalty Points"));
18 VALIDATE("Invoice Discount Amount",
19          "Invoice Discount Amount" + "Loyalty Point Discount");
20
```

Das obige Codestück ist teil des *Sales Header* Tabellenobjekts und validiert die Eingaben unter Nutzung der *Record-API*. Die *Record-API* ist sowohl in C/AL als auch in AL zu finden, und liefert alle nötigen Funktionen um mit Tabellenobjekten zu interagieren. Die hier in 5.2 verwendeten Funktionen werden folgend kurz erläutert:

- GET: [Ok :=] Record.GET([Value] ,...) Liefert einen Record anhand seines Primärschlüssels.
- CALCFIELDS: [Ok :=] Record.CALCFIELDS(Field1, [Field2],...) Errechnet den Wert eines FlowFields. FlowFields sind Felder, die zur Laufzeit berechnet werden und aggregierte Werte einer via Fremdschlüssel verbundenen Tabelle liefert. Im obigen Beispiel wird dadurch der aktuelle Treuepunktsaldo des Kunden berechnet. Dieser setzt sich aus der Summe der Werte der Tabelle *Loyalty Point Entry* zusammen.
- VALIDATE: Record.VALIDATE(Field [, NewValue]) Validiert den momentanen Wert eines Feldes anhand der im Tabellenobjekt für das Feld defnierten Validierungslogik. Sollte der Parameter *NewValue* mitgegeben werden, erfolgt vor der Validierung eine Zuweisung des übergebenen Wertes. Sollte für das zu validierenden Feld keine Validierungslogik definiert sein, entspricht das Verhalten der VALIDATE-Funktion einer Zuweisungsoperation.

AL

Im Gegensatz zu C/AL werden unter AL Änderungen nicht direkt im bestehenden Gesamtsystem vorgenommen, sondern werden in abgekoppelte kleine Einheiten verpackt - Erweiterungen. Um eine Erweiterung zu erstellen und veralten zu können, sind einige Metadaten nötig. Diese Metadaten sind im JSON Format in Form einer Datei namens app.json der Erweiterung beizufügen.

Programm 5.2: Metadatendefinition: app.json für die Treuepunkterweiterung

```
1 {
2   "id": "6f30e971-5966-4a04-9dc0-6b4dcbbe3aef",
3   "name": "LoyaltyPoints",
4   "publisher": "Johannes Naderer",
5   "version": "1.0.0.0",
6   "brief": "Extension to manage LoyaltyPoints for Customers",
7   "description": "Delivers configurable functionality for customers to earn and
      spend Loyalty Points, as well as Exports and Reporting capabilities to serve BI
      needs",
8   "dependencies": [],
9   "screenshots": [],
10  "platform": "14.0.0.0",
11  "application": "14.0.0.0",
12  "idRanges": [
13  {
14    "from": 50200,
15    "to": 50249
16  }
17  ],
18  "contextSensitiveHelpUrl": "https://localhost/LoyaltyPoints/",
19  "showMyCode": true,
20  "runtime": "3.0"
21 }
22
```

Neben den gezeigten Eigenschaften stehen in der app.json noch einige andere zur Verfügung. Die wichtigsten Eigenschaften sind jedoch die generierte eindeutige *id* der Erweiterung im GUID-Format, der *name*, die Abhängigkeiten zu anderen Erweiterungen (*dependencies*). Denn anders als unter C/AL können erweiterungsbasierte Konstrukte aufeinander aufbauen. Dementsprechend kann eine Erweiterung beispielsweise Felder zu einem Tabellenobjekt hinzufügen, das von einer anderen Erweiterung erstellt wurde.

Um es Marketingmitarbeitern unseres Auftraggebers zu ermöglichen, einzelnen Kunden zu Marketingzwecken Treuepunkte zu schenken, wird die Debitorenkarte um eine entsprechende Aktion erweitert. Der Nutzer dieser Aktion soll über die in Business Central enthaltene Standard-Filterfunktionalität anhand von Filterkriterien eine Kundenliste zusammenstellen. Über drei weitere für den Benutzer verfügbare Felder sind die zu vergebenen Punkte, die verwendete Belegnummer, und eine Beschreibung zu

vergeben. Nach der Eingabe der entsprechenden Informationen soll nach Bestätigung durch den Benutzer der Prozess zur Vergabe der Treuepunkte anhand der konfigurierten Parameter gestartet werden. Aufgaben wie diese, in denen es einerseits nötig ist, vom Benutzer Konfigurationen abzufragen, und diese andererseits direkt innerhalb der Geschäftslogik zu verarbeiten, werden im Business Central Umfeld bevorzugt mithilfe von Berichtsobjekten umgesetzt. Denn im Gegensatz zu Codeunit-Objekten haben Berichte die Eigenschaft, via grafischer Oberfläche (*Request Page*) konfigurierbar zu sein. Da der Bericht zur Vergabe von Treuepunkten keinen Ausdruck liefern soll, wird in den Eigenschaften des Berichtsobjekts die Einstellung *ProcessingOnly = true* getroffen. Neben trivialen Aufgaben, wie jener der Vergabe von Treuepunkten, werden Berichte dieser Art Systemweit für eine Vielzahl komplexer Aufgaben verwendet. Dazu zählen unter anderem die Regulierung der Artikeleinstandspreise und die Übernahme von Buchungen aus der Finanzbuchhaltung in die Kostenrechnung, beides Prozesse die für die Gesamtfunktionalität des Systems von größter Bedeutung sind.

Mithilfe der vom Berichtsobjekt zur Verfügung gestellten *Request Page*, und den darauf befindlichen Optionen und Filtermöglichkeiten, lassen sich schnell und einfach komplexe Auswahlen tätigen. Sei die Aufgabe, allen Kunden in Oberösterreich innerhalb eines gewissen Kundennummernbereichs, die einen Treuepunktsaldo von 5000 nicht übersteigen, 50 Treuepunkte gut zu schreiben, so ließe sich dies auf folgende Weise erledigen:

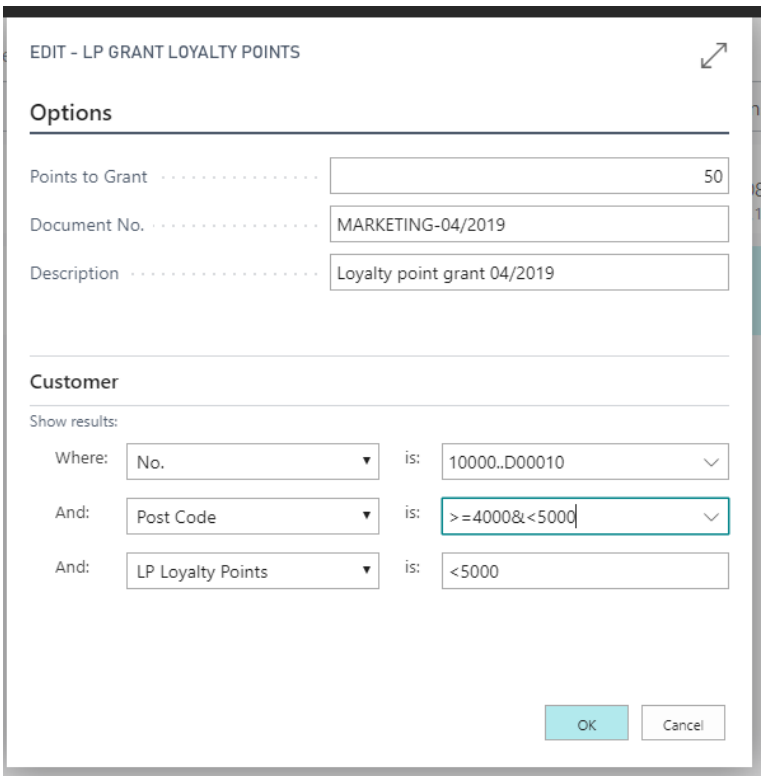

Abbildung 5.6: Grundfunktionalität: Business Central Web Client Request Page, Parametrisierung und Filterkriterien zur Treuepunktvergabe

5.2.3 Datenträgerexport

C/AL

Datenträgerexports werden unter Microsoft Dynamics 365 Business Central vorwiegend mithilfe der XMLPort-Applikationsobjektart durchgeführt. Wie auch bei den bereits vorgestellten Tabellenobjekten ist im Development Environment auch für diese Objekte ein Designer Fenster verfügbar. Im XMLPort Designer lässt sich die Datenstruktur des Exports anhand einzelner Datenelemente in einer grafischen Benutzeroberfläche definieren. Diese Datenelemente stammen entweder direkt aus einem Feld eines Tabellenobjektes, und somit direkt aus der Datenbank, oder können unter Verwendung von Textvariablen zur Laufzeit des XMLPort-Objekts erstellt und manipuliert werden. Um Variablen zu setzen und zu definieren, liefern XMLPort-Objekte eine Vielzahl vordefinierter *Trigger* die in einer fest gesetzten Reihenfolge gefeuert werden. Innerhalb dieser *Trigger* kann der C/AL Code platziert werden, um die zu exportierenden Textvariablen zu manipulieren.

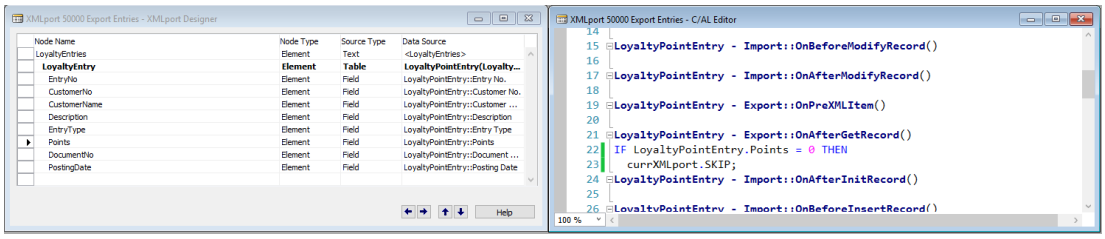

Abbildung 5.7: Grundfunktionalität: Business Central Web Client Request Page, Parametrisierung und Filterkriterien zur Treuepunktvergabe

Innerhalb der definierten Trigger, steht es dem Entwickler frei Code zu platzieren, hier ist jeglicher C/AL Code gültig. So lassen sich die auf den ersten Blick sehr einfach gehaltenen Elemente aus dem XMLPort-Designer auf verschiedenste Arten manipulieren, um auch komplexe Import- und Exportaufgaben zu lösen. Der Export der Treuepunkteposten stellt keine Anforderungen an ein komplexes Datenformat, hier reicht einzig die Definition der zu exportierenden Datensätze im XMLPort-Designer Fenster um an das gewünschte Ziel zu kommen.

AL

In Bezug auf die Möglichkeiten zum Datenträgerexport unterscheiden sich C/AL und AL nur in wenigen, aber enorm wichtigen Aspekten. Während im Development Environment der XMLPort Designer eine einfache Lösung liefert, die Form des Exportdatenträger zu definieren, ist diese Definition unter AL in textueller Form zu treffen. Der wirkliche Unterschied zwischen den den beiden Entwicklungsparadigmen ist jedoch, wie XMLPort-Objekte im System verwendet werden können. In den OnPremise Varianten von Microsoft Dynamics NAV und Microsoft Dynamics 365 Business Central werden die Eigenschaften des XMLPort-Objektes gerne genutzt, um Dateien zu generieren und diese automatisiert im Dateisystem der Servermaschine abzulegen. Mithilfe von XMLPort-Objekten lassen sich jedoch auch Daten von lokal erreichbaren Dateien importieren. Diese Eigenschaften machen es verhältnismäßig einfach, Daten für Drittsysteme zu exportieren, oder mittels Dateisystem Daten aus Drittsystemen zu importieren.

In einer Cloud-Umgebung, in der Dynamics 365 Business Central in der SaaS Konfiguration betrieben wird, steht jedoch aufgrund der Infrastrukturverhältnisse kein Dateisystem zur Verfügung. Der einfache dateibasierte Ansatz ist hier daher fehl am Platz. In der serverlosen Variante wird rein Anhand von Datenströmen (*Streams*) gearbeitet, die im Hauptspeicher der Servermaschine verwaltet werden. Das umschifft zwar das Problem des fehlenden Dateizugriffs, stellt Entwickler aber vor neue Probleme. In einem Anwendungsfall, in dem der Benutzer eine einzelne Datei Importieren möchte, besteht keine Schwierigkeit. Der Nutzer wird vom System nach der Datei gefragt, die gewählte Datei wird auf den Server geladen und in einen für Business Central geeigneten Datenstrom übertragen.

In einem Szenario, in dem eine Vielzahl von Dateien automatisiert ohne Benutzerinteraktion verarbeitet werden müssen, stößt das System hier an seine Grenzen. Ohne einen Benutzer, der die Dateien manuell hochlädt, hat das Cloud-System keinen Zugriff auf die meist lokal liegenden zu verarbeitenden Daten. Was in einer OnPremise Lösung trivial erscheint, wird hier zu einer komplexen Aufgabe, die einige Überlegungen notwendig macht, und meist darin endet, dass ein Cloud-Datenspeicher gekauft werden muss, in den die zu verarbeitenden Daten synchronisiert werden müssen.

Zu den genannten Einschränkungen kommt noch dazu, dass sich XMLPort-Objekte im WebClient nur ohne *Request Page* aufrufen lassen, was ihre Nützlichkeit für den Endbenutzer bei Exportaufgaben deutlich einschränkt, und bei einigen Aufgaben dazu führt, dass ein Export mittels XMLPort-Objekt in der verwendeten Online-Version von Dynamics 365 Business Central (*Spring Release 2019*) schlichtweg keine gute Lösung darstellt.

5.2.4 Reporting

Um grafisch ansprechende Auswertungen über große Datenmengen zu ermöglichen, liefert Microsoft Dynamics 365 Business mehrere Möglichkeiten. Einige der mächtigsten Möglichkeiten befinden sich hier jedoch nicht im System selbst, sondern sind gelungene Anbindungen an andere Systeme und Programme. Bis auf einige wenige Ausnahmen, lassen sich die Daten jeder Ansicht in Business Central als Excel-Dokument exportieren. Dies klingt vorerst nicht besonders Aufregend, liefert Nutzern jedoch die Möglichkeit mit einem bekannten Werkzeug schnell benutzerdefinierte Auswertungen zu erstellen. Neben diesen Exportmöglichkeiten bietet Microsoft Dynamics 365 Business Central auch die Möglichkeit Daten mithilfe vorgefertigter Routinen an das Business Intelligence Systeme *PowerBI* zu synchronisieren.

Aber auch innerhalb des Systems finden sich verschiedene Möglichkeiten, Auswertungen zu gestalten. Zum einen lassen sich sogenannte *Benutzerdefinierte Diagramme* erstellen. Diese Diagramme können durch den Endbenutzer selbst gefertigt werden, und nach belieben in der Startseite des Benutzers angezeigt werden. Somit hat der Benutzer die Möglichkeit, sich je nach seiner Rolle und seinen Aufgaben entsprechende Diagramme zu erstellen, um ihn bei seiner täglichen Arbeit zu unterstützen.

Die meistgenutzte Möglichkeit für grafische Auswertungen und Datenauflistungen stellen jedoch die klassischen, druckbaren Berichte dar. Berichtsobjekte sind durch Entwickler zu erstellen, können jedoch vom Benutzer durch zusätzliche Layouts erweitert werden. Wie auch in XMLPort-Objekten liefert auch der Report-Designer im Development Environment eine intuitive Möglichkeit eine Datendefinition zu erstellen. Wie bei allen anderen Objektarten unter C/AL liefert auch der Bericht verschiedenste Trigger, die es dem Entwickler ermöglichen Code zu platzieren, und so auf das Verhalten des Berichts zur Laufzeit Einfluss auszuüben. Das Laufzeitverhalten eines Berichtsobjekts teilt sich in zwei Unterteile auf, die nacheinander ausgeführt werden. Zuerst wird anhand der Datendefinition und des hinterlegten Codes ein Dataset erstellt. Bei diesem Dataset handelt es sich um einen normierten XML-Datenträger. Dieses Dataset wird im zweiten Schritt in ein Layout übertragen, dass die im Dataset enthaltenen Daten entsprechend visuell aufbereitet. Bei diesen Layouts unterscheidet man im Business Central Umfeld zwischen RDLC und Word-Layouts. Bei RDLC (*Report Definition Language Client-side*) handelt es sich um einen XML-Dialekt, der ein grafisches Berichtslayout beschreibt. Üblicherweise werden RDLC-Layouts mit einem grafischen Editor erstellt. Unter Windows geschieht dies üblicherweise mit Microsoft Visual Studio. Von Entwicklern erstellte RDLC-Layouts können durchaus komplex werden, da es innerhalb von RDLC-Layouts möglich ist mithilfe der Sprache VisualBasic Anzeigelogik zu definieren. Mithilfe von Word-Layouts lassen sich Datasets aber auch vom Benutzer selbst in druckbare Form bringen. Microsoft Word liefert hier einige Werkzeuge, mit denen grafische Berichtslayouts anhand von XML-Definitionen erstellt werden können. Die aus Business Central erzeugten Datasets entsprechen genau diesen XML-Definitionen. Fertig gestellte Word-Layouts werden nach vollendung auf den Business Central Server geladen, und können so für die grafische Repräsentation von Datasets anstatt der standardmäßig hinterlegten RDLC-Layouts verwendet werden.

5.2.5 Webservice-Anbindung

C/AL

Im letzten Entwicklungsschritt soll die ansonsten komplettierte Erweiterung zur Verwaltung von Treuepunkten an das CRM-System des Kunden angebunden werden. Diese Anbindung passiert über einen vom CRM-System veröffentlichten REST Webdienst, über den einzelne Treuepunkteposten übermittelt werden können. Neben diesem einfachen Webdienst sind für die Zukunft noch eine Vielzahl weiterer, komplexerer Schnittstellen geplant.

C/AL und die in der Sprache inkludierten Funktionen reichen für die Erfüllung der geforderten Aufgabe nicht aus. In C/AL ist keine Möglichkeit vorgesehen, mit Drittsystemen via HTTP zu kommunizieren. Um der Anforderung trotzdem gerecht zu werden, wird eine Funktionalität genutzt, die mit der Version 2009R2 in das System integriert wurde: DotNet Interoperabilität. Mit Microsoft Dynamics NAV 2009R2 wurde die 3-Schichten-Architektur eingeführt, und mit ihr eine Serverschicht, die auf .NET Basis entwickelt wurde, und auch .NET Code ausführt. Entwicklern steht damit offen .NET basierte Klassenbibliotheken zu erstellen, in das bestehende System zu integrieren, und Funktionalität aus diesen Bibliotheken aus C/AL aufzurufen. Obwohl mit der Einführung von .NET in C/AL die Möglichkeiten unbegrenzt erscheinen, ist die Einbindung von .NET Klassenbibliotheken nur sehr begrenzt sinnvoll. Ein Grund hierfür ist die Abwesenheit einer wohl definierte Anbindung an das Common Type System aus dem .NET Umfeld. Die Kommunikation zwischen den beiden Welten kann nur über einfache Datentypen passieren, da C/AL nicht mit Objekten aus einem objektorientierten Umfeld umgehen kann, und C/AL Records nicht an die .NET Klassenbibliothek übergeben werden können. Weiters können einige Features des .NET Frameworks wie generische Typen und Collections nicht, oder nur via Reflection in C/AL verwendet werden.

Um Treuepunkteposten an den Webdienst senden zu können, wird anhand fehlender Alternativen in C/AL eine .NET Komponente *WebRequestHandler* entwickelt, welche die Datenkonvertierung und HTTP-Kommunikation übernimmt. Im folgenden wird für die Implementierung die Sprache C# verwendet, hierfür können jedoch sämtliche .NET Sprachen verwendet werden, unter anderen also auch VisualBasic oder F#.

Programm 5.3: .NET Interop: C# Code zur Übermittlung eines Treuepunktpostens an den CRM-Webdienst

```
1  public class WebRequestHandler
2  {
3    private int TransmitEntry(LoyaltyPointEntryDTO dto, string endpoint)
4    {
5      try
6      {
7        HttpClient client = new HttpClient();
8        client.BaseAddress = new Uri(endpoint);
9        client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("
       application/json"));
10       HttpContent content = new StringContent(dto.ToJsonString(), Encoding.UTF8, "
       application/json");
11
12       HttpResponseMessage response = client.PostAsync(endpoint, content).Result;
13       return (int)response.StatusCode;
14     }
15     catch (Exception e)
16     {
17       return 0;
18     }
19   }
20
```

Die Methode *TransmitEntry* 5.3 empfängt hier ein Objekt vom Typ *LoyaltyPointEntryDTO*, dass bereits sämtliche Felder der Business Central Tabelle als Eigenschaften enthält. Danach wird mittels der vom .NET Framework zur Verfügung gestellten Klasse HttpClient eine entsprechende Webanfrage erstellt und an den Webdienst übermittelt. Bei der Implementierung von .NET Code, der aus C/AL aufgerufen werden soll, sind gegenüber der traditionellen Entwicklung von .NET Applikationen einige Punkte zu beachten:

- An die aufrufende C/AL Komponente können lediglich einfache Typen zurückgegeben werden, und selbst hier können bei Gleitkommazahlen unter gewissen Umständen ungewollte Nebeneffekte auftreten. Es empfiehlt sich, wenn möglich nur Ganzzahlen und Zeichenfolgen als Rückgabewerte zu verwenden. Hier wird der HTTP Status Code verwendet, um C/AL rückzumelden, ob der Aufruf erfolgreich war.
- Asynchrone Aufrufe sind nicht möglich. Die Laufzeitumgebung von Microsoft Dynamics Business Central führt C/AL immer synchron aus. Obwohl Netzwerkkommunikation über HTTP grundsätzlich eine Operation ist, die asynchron ausgeführt werden sollte, wird die C/AL Umgebung immer auf das Ausführungsende warten.
- Sämtliche Ausnahmen (*Exceptions*) müssen im .NET Code behandelt werden. Microsoft Dynamics Business Central liefert nur sehr begrenzt Möglichkeiten mit Ausnahmesituationen umzugehen, es empfielt sich daher Ausnahmen dort zu behandeln wo sie auftreten, und das Auftreten einer Ausnahme über den Rückgabewert an die aufrufende C/AL Komponente zu signalisieren.

- C/AL liefert keinen Mechanismus, um im Web beliebte Formate wie *JSON* (JavaScript Object Notation) zu verarbeiten. Die Serialisierung und Deserialisierung von Daten sollte somit im .NET Code implementiert werden. Im obigen Codebeispiel passiert dies über die Methode *ToJsonString()* der Klasse *LoyaltyPointEntryDTO*.

Um die fertige Klassenbibliothek schlussendlich auch in C/AL verfügbar zu machen, muss die kompilierte Bibliothek im nächsten Schritt entweder im *Global Assembly Cache* der Servermaschine registriert, oder in das Programmverzeichnis der Serverapplikation kopiert werden. Danach kann im Development Environment eine Variable vom Typ DotNet und Subtyp der erstellten Klassenbibliothek erstellt werden, über die die Methoden der Klassenbibliothek aufgerufen werden können.

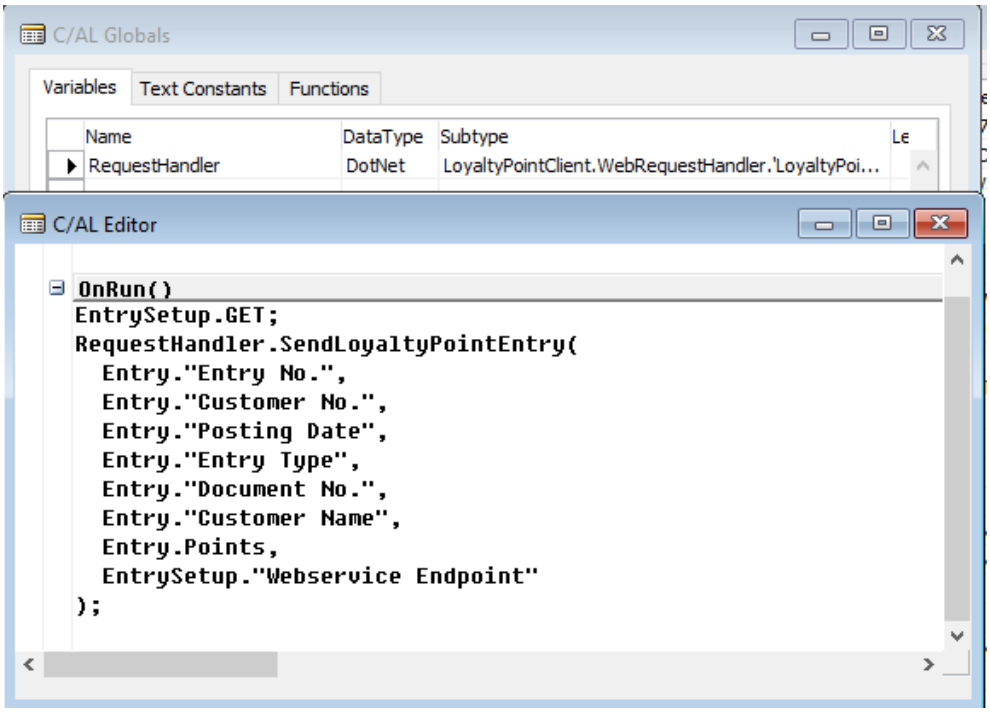

Abbildung 5.8: Development Environment: Nutzung eines DotNet Datentypen zur Kommunikation mit einem CRM-Webdienst

AL

Anders als im Development Environment und unter C/AL lässt sich eine Webdienstanbindung mithilfe der Möglichkeiten in AL ohne dem Umweg über das .NET Framework und die DotNet Interoperabilität realisieren. In der OnPremise Variante von Microsoft Dynamics 365 Business Central wird die Verwendung von selbst implementierten Klassenbibliotheken zwar noch unterstützt, sobald man sich jedoch einer Cloud-Instanz von Business Central bewegt, ist diese Möglichkeit nicht mehr verfügbar. Mit dem Wegfall von DotNet Interoperabilität in der Cloud-Variante geht auch viel potentielle Funktionalität der Lösung verloren. Um dem entgegenzuwirken, wurde und wird die Sprache AL

auch in Zukunft immer weiter um Funktionalität angereichert. Beispiele dieser neuen Funktionalitäten in AL sind unter anderem die neuen Datentypen zur Kommunikation via HTTP und zur Erstellung von JSON und XML Dokumenten. Neben diesen ist es unter AL auch erstmals in der Geschichte von Dynamics 365 Business Central möglich, im Speicher verwaltete *Collections* zu nutzen, die nicht auf temporären Datenbanktabellen basieren. Dazu ist anzumerken, dass diese neuen Funktionalitäten nicht in AL implementiert sind, sondern lediglich eine syntaktische Erweiterung darstellen. Da sowohl C/AL als auch AL Code vor dessen Ausführung in einen C# - Zwischencode transpiliert, und darauf in *MSIL - Microsoft Intermediate Language* kompiliert wird, sind die zur Laufzeit exekutierten Klassen und Algorithmen immer jene aus dem .NET Framework.

Nichts desto trotz bietet die Erweiterung von AL um die neuen Komponenten Entwicklern bisher nicht vorhandene Möglichkeiten. Mithilfe der Http-Datentypen und der *Dictionary* Collection kann in AL ohne großen Aufwand, und ohne Umwege über eine andere Programmiersprache eine Funktion 5.4 erstellt werden, die beliebige Http-Webanfragen erstellen und absenden kann. *DoPostRequest* empfängt eine Endpunktadresse, zwei Behälter (*Collections*) für nötige Kopf-Metadaten und den Anfragetext *Http-Body*. Im Anschluss wird aus den Eingabedaten eine *HttpRequestMessage* erstellt, die aus zwei *HttpHeader* Komponenten und einem *HttpContent* besteht. Der Datentyp *HttpClient* liefert eine einfache Methode *Send()*, welche die defnierte Http-Anfrage absendet, und die Antwort in Form eines Ausgabeparameters *Response* bereitstellt.

Programm 5.4: AL: Helfer Funktion zur Übermittlung von POST-Anfragen an Webdienste

```
procedure DoPostRequest(Endpoint: Text;
                        RequestHeaders: Dictionary of [Text, Text];
                        ContentHeaders: Dictionary of [Text, Text];
                        Body: Text;
                        var Response: HttpResponseMessage)
var
  Client: HttpClient;
  Request: HttpRequestMessage;
  Headers: HttpHeaders;
  Content: HttpContent;
  DictKey: Text;
begin
  Request.Method := 'POST';
  Request.SetRequestUri(Endpoint);
  Request.GetHeaders(Headers);

  foreach DictKey in RequestHeaders.Keys() do
    Headers.Add(DictKey, RequestHeaders.Get(DictKey));

  Content := Request.Content();
  Content.WriteFrom(Body);

  Content.GetHeaders(Headers);
  foreach DictKey in ContentHeaders.Keys() do begin
    Headers.Remove(DictKey);
    Headers.Add(DictKey, ContentHeaders.Get(DictKey));
  end;
  Request.Content(Content);
  Client.Send(Request, Response);
end;

```

Um die Hilfsfunktion korrekt zu nutzen, und eine syntaktisch korrekte HTTP-Anfrage abzusenden, ist es nötig, der Funktion valide Kopf-Metadaten und einen zum Dienst passenden Inhalt zu übermitteln. Der Webdienst Endpunkt der CRM-Lösung erwartet Eingabedaten im JSON Format. Um die Daten aus Microsoft Dynamics 35 Business Central entsprechend aufzubereiten, wurden in AL gleich mehrere Datentypen zur Erstellung und Auswertung von JSON Dokumenten eingeführt:

- JsonObject: Repräsentiert ein Objekt in der Json Notation.
- JsonArray: Repräsentiert ein Feld von Json Tokens.
- JsonToken: Repräsentiert ein Json Element, also entweder in Objekt, ein Feld, oder einen einfachen Wert.
- JsonValue: Repräsentiert einen einfachen Datenwert, wie eine Zahl oder eine Textfolge.

Mithilfe dieser vier Datentypen lassen sich beliebig komplexe und tief verschachtelte Json Dokumente erstellen und verarbeiten. Um einen Treuepunktposten an den Web-

dienst zu übertragen, sind jedoch keine komplexen Strukturen nötig, die Verwendung von JsonObject reicht für diesen Anwendungsfall aus. 5.5 Die Prozedur *SendRequests* empfängt eine vorgefilterte Record-Variable von Treuepunktposten, und erstellt für jeden Posten ein neues JSON-Dokument *EntryObject*, welches in weiterer Folge mithilfe der Hilfsprozedur *DoPostRequests* 5.3 an den Webdienst übermittelt wird.

Programm 5.5: AL: Erstellung der Grundstruktur der Http-Anfrage zur Übermittlung von Treuepunktposten.

```
1 local procedure SendRequests(endpoint: Text; var LoyaltyPointEntry: Record "LP
      Loyalty Point Entry"): Text
2 var
3   WebServiceHelper: Codeunit "LP WebServiceHelper";
4   EntryObject: JsonObject;
5   RequestHeaders: Dictionary of [Text, Text];
6   ContentHeaders: Dictionary of [Text, Text];
7   Body: Text;
8   ResponseMessage: HttpResponseMessage;
9 begin
10  ContentHeaders.Add('Content-Type', 'application/json');
11  With LoyaltyPointEntry do
12    if FindSet() then
13      repeat
14        Clear(EntryObject);
15        Body := '';
16        EntryObject.Add('entryNo', "Entry No.");
17        EntryObject.Add('customerNo', "Customer No.");
18        EntryObject.Add('postingDate', "Posting Date");
19        EntryObject.Add('entryType', Format("Entry Type"));
20        EntryObject.Add('documentNo', "Document No.");
21        EntryObject.Add('customerName', "Customer Name");
22        EntryObject.Add('points', Points);
23        EntryObject.WriteTo(Body);
24        WebServiceHelper.DoPostRequest('http://.../api/LoyaltyEntryPoint',
25                                       RequestHeaders,
26                                       ContentHeaders,
27                                       Body,
28                                       ResponseMessage);
29      until LoyaltyPointEntry.Next() = 0;
30 end;
31 }
32
```

Kapitel 6

Tests und Evaluierung

6.1 .NET Klassenbibliothek und AL-Sprachkonstrukte

6.1.1 Testaufbau

Im ersten Testaufbau werden die im Treuepunkt-Beispiel 5.1 verwendeten Varianten der Webservice Kommunikation gemessen und auf ihr Laufzeitverhalten untersucht. In diesem Test werden beide Varianten eine große Anzahl von Http-Anfragen erstellen und an einen Webdienst übermitteln. Um die Ergebnisse vergleichbar zu halten, muss in beiden Varianten dasselbe JSON erstellt und übermittelt werden. Bei einer Messung dieser Art sind grundsätzlich immer hohe Schwankungen zu erwarten, da sowohl die momentane Netzwerklast, als auch die Verarbeitungszeit des Webdienstes mitgemessen wird. Um diese Seiteneffekte zu minimieren, beziehungsweise sie konstant zu halten, wird in diesem Test an einen eigens Erstellten Webdienst übertragen. Dieser mit ASP.NET Core und die ausführende Microsoft Dynamics 365 Business Central Instanz laufen auf derselben Maschine. Dadurch entfällt die Messungenauigkeit durch Netzwerklatenzen. Um die Bearbeitungszeit des Webdienstes möglichst gering zu halten, werden auf dieser Seite keine Operationen durchgeführt. Es werden lediglich die Eingangsdaten auf Ihre syntaktische Richtigkeit überprüft. Mit diesen beiden Maßnahmen in Platz fallen die sonst vergleichsverfälschenden Störungen weg und haben so keine bedeutende Auswirkung auf die Messergebnisse. Vor dem Start der Testläufe werden sowohl Datenbank als auch Serverdienste neu gestartet, sodass für den ersten Testlauf der Testreihe jeweils ein sauberes System ohne zwischengespeicherte Objekte zur Verfügung steht.

Als Messwerkzeug wird das in Microsoft Dynamics 365 Business Central enthaltene sogenannte *TestTool* verwendet, das grundsätzlich zur Ausführung von automatisierten Tests genutzt wird. Da dieses Testwerkzeug auch präzise Laufzeitmessungen vornimmt, und sämtliche durch den getesteten Code verursachten Nebeneffekte mitmisst, stellt es sich für diesen Anwendungsfall als optimales Werkzeug dar.

6.1.2 Messungen

Tabelle 6.1: Ergebnisse für 10000 Anfragen, Messwerte in Sekunden

	Testlauf 1	Testlauf 2	Testlauf 3	Testlauf 4	Durchschnitt	Std. Abweichung
C/AL .NET	45,96	47,174	43,707	44,28	45,280	1,371
AL	37,247	37,396	37,166	38,016	38,016	0,334

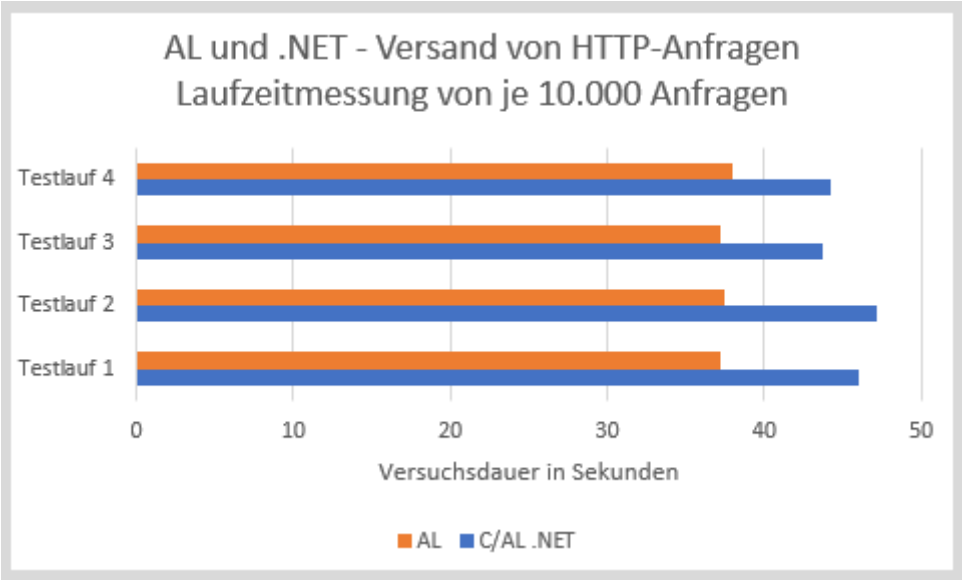

Abbildung 6.1: Grafische Darstellung der Messergebnisse der Laufzeitmessungen

Die Messungen zeigen wie erwartet, dass die in die Sprache integrierte Funktionalität ein leicht besseres Laufzeitverhalten zeigt. Trotz des Mehraufwandes des Ladens einer externen Klassenbibliothek und deren Verwendung, verhält sich die AL-Variante jedoch im Durchschnitt lediglich 16% performanter.

6.2 Tabellenanpassung und Tabellenerweiterung

Wie im Unterkapitel Objektarten in Business Central 4.3 erwähnt, können in der erweiterungsbasierten Programmierung unter AL zu bestehenden Tabellenobjekten keine neuen Felder direkt hinzugefügt werden. Um dennoch die Möglichkeit zu wahren, die Standardtabellen zu erweitern, wurde der Objekttyp der Tabellenerweiterung eingeführt. In Objekten vom Typ Erweiterungstabelle platzierte Felder werden in der Datenbank ein einer eigenen Tabelle abgelegt. Diese zusätzliche Tabellen sind unbedingt nötig, um Daten der Erweiterung von denen der Standardapplikation zu kapseln, und so zu garantieren, dass die Standardapplikation ohne manuelle Eingriffe mit Updates versorgt werden kann. Gleichzeitig hat dieses Verhalten jedoch auch zur Folge, dass der Zugriff auf die zusätzlichen Felder mit einer *Join-Operation* erfolgen muss. In weiterer Folge können diese Felder auch nicht als *SumIndexField* in einem Tabellenindex verwendet werden kann.

Ziel der folgenden Testreihe ist es, den Unterschied zwischen einer traditionell zum Tabellenobjekt hinzugefügten Feld, und einem via Erweiterungsobjekt erstelltem Feld zu quantifizieren. Zu diesem Zweck wird die C/AL Variante der Tabelle *Loyalty Point Entry* 5.4 mittels einer neuen Erweiterung um ein neues Feld *Points2* erweitert, sodass sich folgendes Tabellenschema ergibt:

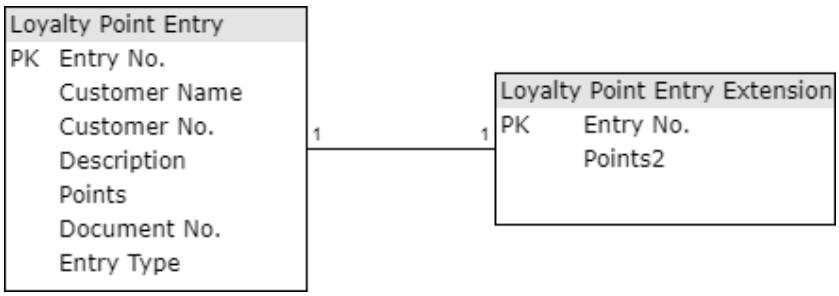

Abbildung 6.2: Tabellenschema Treuepunktposten und zugehörige Erweiterung

Um die Abfrage um herauszufinden, wie viele Treuepunkte ein Kunde momentan hat, möglichst effizient gestalten zu können, wird in der C/AL Variante - also dem Feld *Points* ein nach *Customer No.* gruppierter Index erstellt. Da *Points2* sich auf Datenbankseite nicht in der selben Tabelle befindet, lassen sich derartige Zugriffsoptimierungen via Indizes hier nicht verwirklichen. Für den folgenden Test werden für zehn Kundeneinträge je 10.000 Treuepunktposten erstellt. Gemessen wird die Zeit, welche die Microsoft Dynamics 365 Business Central Instanz benötigt, um für jeden der Kunden den Treuepunktsaldo zu ermitteln. In dieser Messreihe wird das Gesamtsystem (Datenbank + Serverinstanz) nach jeder Messung neu gestartet, um Effekte durch Zwischenspeicherungen auszunehmen.

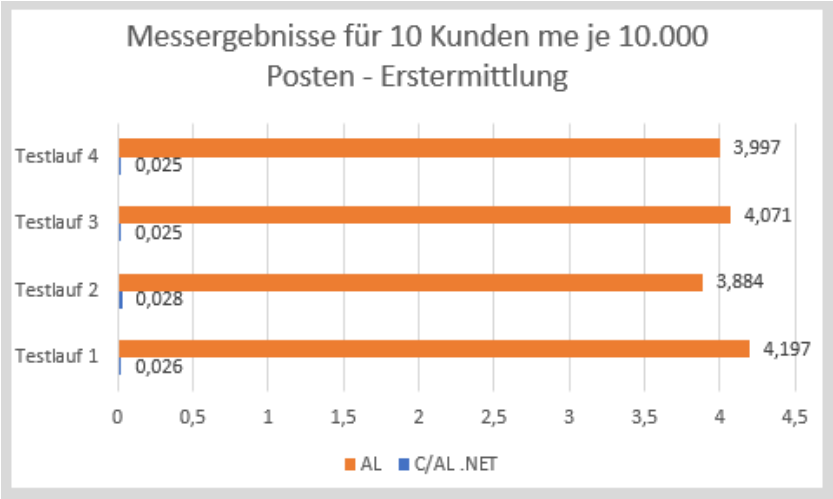

Abbildung 6.3: Messung: Aggregierung von Treuepunktposten - Ohne Zwischenspeicher

Hier lässt sich ein deutlicher Unterschied in Bezug auf die Berechnungszeiten feststellen. Die Differenz rührt daher, da das via C/AL erstellte Feld auch gleichzeitig von Microsoft Dynamics 365 Business Central als *SumIndexField* angelegt wurde. Anhand der angegebenen *SumIndexFields* kann Microsoft SQL Server materialisierte Sichten erstellen, um Daten bereits beim Einfügen zu aggregieren. Somit wird beim Zugriff auf das Feld in der C/AL Variante keine Summenoperation ausgeführt, sondern lediglich das Ergebnis aus der materialisierten Sicht wiederverwendet, währenddessen muss für die Auswertung des Feldes *Points2* aus dem Erweiterungstabellenobjekt jeder Treuepunktposten mit seinem Äquivalent aus der Erweiterungstabelle verbunden werden, und danach die Summe der einzelnen Werte berechnet werden. Dies würde bedeuten, dass jede Berechnung den Nutzer einige Sekunden an Zeit kosten würde. In dieser Testreihe werden lediglich 10.000 Datensätze verwendet, in einem Produktivszenario sind jedoch oft wesentlich größere Datenquellen vorhanden. Um diese Probleme zu mindern, verwendet Microsoft Dynamics 365 Business Central ein komplexes Zwischenspeichersystem, das oft verwendete Datenbestände im Hauptspeicher verwaltet und vor-aggregiert. Vor der folgenden Messung wurde das System nicht durchgestartet. Die Serverapplikation hat hier bereits Treuepunktdaten im Hauptspeicher vorberechnet. Dies ist der Zustand in dem sich das System im Normalzustand befindet.

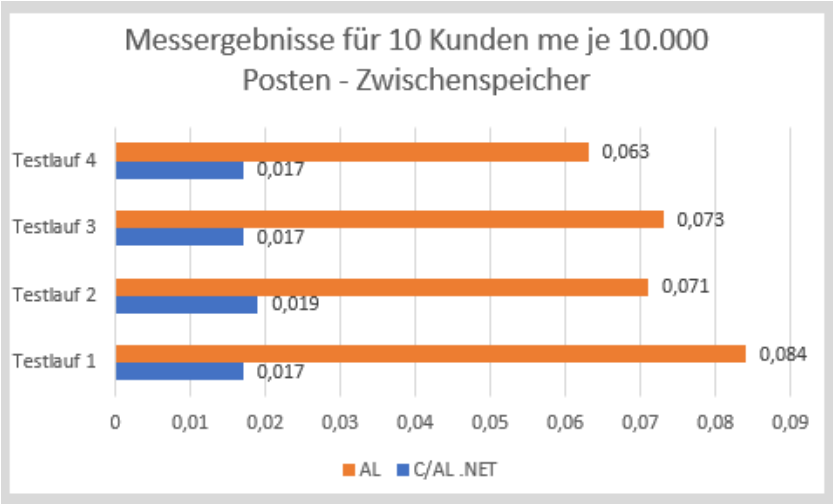

Abbildung 6.4: Messung: Aggregierung von Treuepunktposten - mit Zwischenspeicher

Der Zugriff auf die relevanten Daten dauert in diesem Zustand zwar immer noch etwas vier mal länger, Berechnungszeiten von unter 100 Millisekunden sind jedoch bei Operationen dieser Art durchaus vertretbar.

6.3 Codeanpassung und ereignisorientierte Codeerweiterung

Mit der Einführung der Erweiterungsentwicklung und AL, werden den Entwicklern rund um die Welt viele neue Möglichkeiten präsentiert. Einerseits ändern sich mit dem erweiterungsbasierten Ansatz Lizenz- und Finanzierungsmodelle, andererseits werden dadurch Entwicklern neue Vertriebsmöglichkeiten zugänglich gemacht. Erweiterungen können ohne mitwirken des Entwicklers beim Endkunden, und durch den Endkunden selbst installiert werden, da die Integration einer Erweiterung laut technischer Spezifikation keine manuell auszuführenden Tätigkeiten bedingt.

Andererseits werden Entwickler durch diesen Umschwung jedoch auch technologisch eingeschränkt. Anpassungen des Microsoft-Standard Codebestands waren seit Ersterscheinung des ERP-Systems der einzige und bevorzugte Weg, um Änderungen am Verhalten des Systems vorzunehmen. Diese Codeanpassungen geben dem Entwickler vollständige Kontrolle über den Ablauf der einzelnen Teilprozesse im Gesamtsystem. So können Entwickler Programmcode an beliebigen Stellen hinzufügen, bearbeiten oder auch entfernen.

Mit der Einführung der erweiterungsorientierten Entwicklung sind Codeanpassungen nicht mehr möglich. Standard-Applikationsobjekte können nicht mehr verändert werden, lediglich Erweiterungen sind noch möglich. Dies würde jedoch auch bedeuten, dass Entwickler nicht mehr die Möglichkeiten hätten, im Microsoft Standard abgebildete Prozesse abzuändern. Um Entwicklern dennoch Anpassungen der Standardlogik zu ermöglichen, wird seitens Microsoft ein Mechanismus eingeführt, der in anderen Sprachen und Plattformen seit Jahrzehnten zum Standardumfang zählt: Ereignisse (*Events*).

Seit dem Ersterscheinen von Ereignissen in der Version 2016, werden in der ge-

samtem Applikation und in sämtlichen Prozesslogiken Ereignisse gefeuert. Entwickler können sich an diese Ereignisse binden und je nachdem welche Ereignisse auftreten, spezielle Logikteile ausführen. Mittlerweile befinden sich in der aktuellen Version von Microsoft Dynamics 365 Business Central mehrere tausend Ereignisse, die aus Codeunit-Applikationsobjekten gefeuert werden.

So erlaubt uns ein Ereignis beispielsweise, darauf zu reagieren, nachdem ein Benutzer beim Drucken eines Berichts einen Drucker auswählt. Das Ereignis hierfür kommt aus dem Codeunit-Applikationsobjekt *ReportManagement*, und mit dem folgenden Codestück kann man auf dieses Ereignis reagieren:

```
1 [EventSubscriber(ObjectType::Codeunit, Codeunit::ReportManagement, '
      OnAfterGetPrinterName', '', false, false)]
2   local procedure OnGetPrinterName(ReportID: Integer; VAR PrinterName: Text[250])
3   begin
4     //Place custom code here
5   end;
6
```

Microsoft stellt mit den gefeuerten Ereignissen in der Codebasis tausende Stellen bereit, um sich in die einzelnen Prozessschritte des Gesamtsystems einzuhaken. Zusammenfassend könnte man meinen, dass dadurch die Notwendigkeit von Codeanpassungen vollständig verschwunden wäre. Dem ist jedoch leider nicht so, denn während Ereignisse eine elegante Möglichkeit bieten, Standardvorgehensweisen im System zu manipulieren, hat auch Ereignismechanismus seine Einschränkungen und Schwächen:

- Was passiert, wenn mehrere Programmstücke auf das selbe Ereignis reagieren wollen? Konkret geht es darum um die Ausführungsreihenfolge. Welcher der reagierenden Codestücke werden zuerst ausgeführt? Da man als Entwickler davon ausgehen muss, dass eine entwickelte Erweiterung auf fremden System ausgeführt wird, ist diese Frage von hoher Wichtigkeit. Denn wenn mehrere Codestücke dieselben Daten ändern, kann es schnell zu Inkonsistenzen kommen. Und inkonsistente Daten können in der Welt der Finanzbuchhaltung schnell zu rechtlichen Schwierigkeiten führen. Bei Änderung eines Systems via Codeanpassung kommt diese Frage nicht auf, die Ausführungsreihenfolge ist strikt durch den Ablauf des Programmcodes gegeben. Auf die Frage, welches Codestück zuerst ausgeführt wird, gibt es keine klare eindeutige Antwort. Im Optimalfall ist der reagierende Code so zu gestalten, dass er weder durch vorhergehende Ereignisbindungen beeinträchtigt wird, noch darauffolgende Ereignisbindungen beeinträchtigt. In sehr vielen Anwendungsfällen ist dies jedoch schlichtweg nicht möglich.
- Die Ereignisarchitektur schränkt Entwickler dramatisch ein. Entwickler können nur noch dort eingreifen, wo ein Eingriff seitens Microsoft auch vorgesehen ist. Und selbst an diesen Punkten kann man Prozesse aus der Basiscode nur beschränkt modifizieren, da man als Entwickler meist keinen Zugriff auf den Zustand aller lokalen und globalen Variablen innerhalb der aufrufenden Funktion hat. Um das vergleichbares Maß an Flexibilität wie Codeanpassungen zu bieten, wäre praktisch nach jeder Zeile Code in der Standardapplikation eine Ereignisauslösung notwendig.

6.4 Sourcecode Verwaltung und CI/CD

Mit der Nutzung von Visual Studio Code haben Microsoft Dynamics 365 Business Central Entwickler erstmals in der Geschichte der ERP-Lösung Zugriff auf einen modernen dateibasierten Quellcode-Editor. Programmcode wird nicht mehr primär in einer Entwicklungsdatenbank bearbeitet und gespeichert, sondern befindet sich lokal auf der Maschine des arbeitenden Entwicklers. Was für Entwickler sämtlicher höherer Programmiersprachen bereits seit Anfang der 90er Jahren Standard ist, gilt seit 2018 auch für Business Central Entwickler. Dabei handelt es sich um einen gigantischen Schritt in die richtige Richtung.

Im Rahmen der *Mibuso NAVTechDays* [1] im November 2018 in Antwerpen, der wohl bedeutendsten Entwicklerkonforenz des Jahres für NAV und Business Central Entwickler im europäischen Raum, wurde während eines Vortrags erhoben, wie viele der Anwesenden Entwickler Sourcecode Verwaltungssysteme wie Git für ihre C/AL Projekte verwenden. Die ernüchternde Antwort: nur etwa 10-15% der Anwesenden Entwickler verwenden für C/AL Programmcode ein entsprechendes Verwaltungssystem. Die Gründe hierfür sind naheliegend: Wenn Programmcode nicht bereits während des Entwicklungsprozesses lokal in Dateiform auf dem Entwicklerrechner vorhanden ist, stellt die Nutzung eines Systems wie Git einen Mehraufwand dar, und verkompliziert den Entwicklungsprozess. Hier ist es oft einfacher und zielführender, nächtlich automatisiert Sicherungen der Entwicklungsdatenbank zu erstellen, und diese bei Bedarf wiederherzustellen.

[1] www.mibuso.com

Kapitel 7

Diskussion

Quellenverzeichnis

Literatur

[1] Edward A. Duplaga und Marzie Astani. „Implementing ERP in Manufacturing“. 20 (Juni 2003), S. 68–75. URL: http://www.dsg.univr.it/documenti/Occorrenzalns/matdid/matdid965805.pdf (siehe S. 3).

[2] Dr Bernard Wong und David Tein. „Critical Success Factors for ERP Projects“ (Jan. 2003), S. 1–2. URL: https://www.researchgate.net/publication/229022123_Critical_Success_Factors_for_ERP_Projects (siehe S. 3).

[3] Mark Brummel. *Learning Dynamics NAV Patterns. Create solutions that are easy to maintain, are quick to upgrade, and follow proven concepts and design.* Packt Publishing, 2015, S. 135–136. URL: https://www.microsoft.com/en-us/p/learning-dynamics-nav-patterns/fgqpf3h0qc1j?activetab=pivot%3aoverviewtab (siehe S. 4).

[4] Michaela Gayer. *Microsoft Dynamics NAV - Einführung in Design und Programmierung.* mbst books, 2016 (siehe S. 7).

[5] Jürgen Ebert Michaela Gayer Christian Hauptmann. *Microsoft Dynamics NAV 2018: Das Anwenderbuch zur Abwicklung von Geschäftsprozessen.* Carl Hanser Verlag GmbH Co KG, 2018 (siehe S. 5).

[6] Fiona Fui-Hoon Nah, Silvana Faja und Teuta Cata. „Characteristics of ERP software maintenance: a multiple case study“. *Journal of Software Maintenance and Evolution: Research and Practice* 13.6 (2001), S. 399–414. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.239 (siehe S. 5).

[7] Kay Giza Tobias Kahlert. *Visual Studio Code Tips and Tricks.* Microsoft Press, 2016. URL: https://www.microsoft.com/germany/techwiese/aktionen/visual-studio-code-ebook-download.aspx (siehe S. 15).