



Fachhochschul-Bachelorstudiengang
SOFTWARE ENGINEERING
A-4232 Hagenberg, Austria

Erweiterungsentwicklung für Microsoft Dynamics 365 Business Central unter Verwendung von ExtensionsV2

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science in Engineering

Eingereicht von

Johannes Naderer

Begutachtet von DI (FH) Dr. Gerald Adam Zwettler MSc

Hagenberg, Mai 2019

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Datum

Unterschrift

Inhaltsverzeichnis

1	Einleitung	3
1.1	Motivation	3
1.2	Zielsetzung	4
2	Grundlagen	5
2.1	ERP: Definition und Übersicht	5
2.2	Geschichte und Grundarchitektur Dynamics 365 Business Central	7
2.2.1	Geschichte	7
2.2.2	3-Schichten Architektur	7
2.3	Objektarten in Business Central	9
3	Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung	12
3.1	Aufgabenstellung	12
3.2	Entwicklungsprozess	13
3.2.1	Entwicklungsumgebung	13
3.2.2	Grundfunktionalität	18
3.2.3	Datenträgerexport	23
3.2.4	Reporting	25
3.2.5	Webservice-Anbindung	26
4	Evaluierung & Diskussion	32
4.1	.NET Klassenbibliothek und AL-Sprachkonstrukte	32
4.1.1	Testaufbau	32
4.1.2	Messergebnisse: Erstellung von Http-Anfragen	33
4.2	Tabellenanpassung und Tabellenerweiterung	34
4.3	Codeanpassung und ereignisorientierte Codeerweiterung	36
4.4	Source Code Management und CI/CD	38
5	Zusammenfassung & Ausblick	40
5.1	Zusammenfassung	40
5.2	Ausblick	40
	Quellenverzeichnis	42
	Literatur	42
	Online-Quellen	43

Kurzfassung

Cloud Computing bietet Software Herstellern bisher nicht vorhandene technische Möglichkeiten zur Erstellung von hochskalierbaren Hardware Infrastrukturen. Um diese Möglichkeiten mit der bestehenden Software *Microsoft Dynamics 365 Business Central* zu nutzen, und gleichzeitig die Vorteile des Systems zu wahren, wurde ein neues Programmierkonzept eingeführt. Code-Änderungen, wie sie bisher die Norm waren, sind in einem Cloud-System nicht akzeptabel. Daher wurde das Konzept der erweiterungsbasierten Programmierung mit ExtensionsV2 eingeführt. ExtensionsV2 erlaubt es Entwicklern, die Grundfunktionalität des ERP-Systems zu erweitern, ohne die Stabilität und Updatefähigkeit des Systems negativ zu beeinflussen.

Diese Arbeit vermittelt einen Überblick über das Programmsystem *Microsoft Dynamics 365 Business Central*, der verwendeten Schichtenarchitektur und vergleicht die Programmierkonzepte hinter den beiden Sprachen C/AL und AL.

Nach der Definition des Begriffs *ERP-System* und der geschichtlichen Entwicklung des Systems *Microsoft Dynamics 365 Business Central* werden die verschiedenen Arten von Objekten, die als Grundbausteine des Systems fungieren beleuchtet. Im Hauptteil wird das neue Programmiermodell mit ExtensionsV2 mit der Code-Anpassung unter C/AL verglichen. Um einen quantifizierbaren Vergleich zu ermöglichen wird eine Erweiterung für das System mit beiden Arten der Programmierung umgesetzt. Da beide Implementierungen schlussendlich dieselbe Funktionalität zur Verfügung stellen, kann so speziell auf die Unterschiede zwischen den beiden Programmierparadigmen eingegangen werden. Anschließend werden Laufzeitmessungen durchgeführt, um Differenzen zwischen den Implementierungen aufzudecken. Abschließend werden die Implikationen bezüglich zeitgemäßer Entwicklungswerkzeuge wie Source Code Management und CI/CD diskutiert, die mit dem dateibasierten Entwicklungsprozess in der erweiterungsbasierten Programmierung in Visual Studio Code und AL einher gehen.

Abstract

Cloud computing offers software manufacturers previously unavailable technical possibilities for creating highly scalable hardware infrastructures. To take advantage of these capabilities with the existing software *Microsoft Dynamics 365 Business Central* while maintaining the benefits of the system, a new programming concept has been introduced. Code changes, as previously the norm, are unacceptable in a cloud system. Therefore, the concept of extension-based programming with ExtensionsV2 has been introduced. ExtensionsV2 allows developers to extend the basic functionality of the ERP system without negatively affecting the stability and updating capability of the system.

This thesis provides an overview of the program system *Microsoft Dynamics 365 Business Central*, the used layer architecture and compares the programming concepts behind the two languages C/AL and AL).

Following the definition of the term *ERP system* and the historical development of the system *Microsoft Dynamics 365 Business Central*, the various types of objects that function as the basic building blocks of the system are examined. The main part of this thesis compares the new extension based programming model using the ExtensionsV2 system to the code adaptation with C/AL. To allow a quantifiable comparison, an extension for the system is implemented with both types of programming. Since both implementations ultimately provide the same functionality, it is possible to specifically address the differences between the two programming paradigms. Next, runtime measurements are performed to reveal differences between the implementations. Finally, the implications of contemporary development tools such as source code management and CI/CD are discussed, which go hand in hand with the new file-based development process in extension-based programming in Visual Studio Code and AL.

Kapitel 1

Einleitung

1.1 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 Softwarelösungen 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 hatten. Solche Systeme können mit den meist begrenzten Ressourcen kleiner und mittelständischer Hersteller erfahrungsgemäß nicht im geforderten Umfang 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 auch 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, bleibt meist nur einer von zwei möglichen 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 [11]. Gerade branchenspezifische und insbesondere unternehmensspezifische Prozesse müssen meist erst programmiert und in das System integriert werden. Programmierarbeiten und Änderungen an den standardmäßigen Prozessen 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 dazu, zertifizierten Entwicklern freien Zugang zum Applikationscode zu gewähren[4]. So können Entwickler die gesamte Geschäftslogik des Systems je nach Unternehmensanforderungen nach Belieben abändern und erweitern, in dem sie neuen Programmcode hinzufügen, oder den Standardcode von Microsoft anpassen oder gegebenenfalls auch 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 Microsofts Codebasis 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 updatefähig zu halten, und gleichzeitig ein Entwicklungsmodell zu schaffen, das 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 Art zu entwickeln, 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.

1.2 Zielsetzung

Im Rahmen dieser Arbeit wird ein Überblick über die ERP-Plattform, die sich mittlerweile *Microsoft Dynamics 365 Business Central* nennt, und die Programmierung dieses Systems 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, das auf beide Arten umgesetzt wird. Einerseits wird die Anwendung in der Entwicklungsumgebung C/SIDE mit C/AL entwickelt. Andererseits wird anhand der Aufgabenstellung eine Erweiterung mit ExtensionV2 in Visual Studio 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 primär anhand von Laufzeitmessungen. Zusätzlich wird auch diskutiert, welche Vor- und Nachteile sich durch die nun neue dateibasierte Codeverwaltung hinsichtlich der Einbindung und Nutzung von Source Code Management und Continuous Integration Systemen ergeben. In einem letzten Block wird danach das Eventbasierte 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 2

Grundlagen

2.1 ERP: Definition und Übersicht

ERP Systeme sind umfangreiche kommerzielle Softwaresysteme mit der Kernaufgabe, alle Abteilungen und Prozesse eines Unternehmens soweit wie möglich digital abzubilden [9]. Ein ERP-System stellt für ein Unternehmen somit eine zentrale Verarbeitungs- und Datensicherungsplattform für sämtliche im Unternehmen relevanten Geschäftsdaten bereit. Die Daten sind hierbei in einem einzelnen System erfasst und sind so sofort für alle Unternehmensbereiche verfügbar. Es sind keine Synchronisierungsschritte oder Schnittstellen innerhalb des Systems nötig. Durch die Datenerfassung in einer zentralen Datenbank stellt ein ERP-System eine durchgängige Informationsquelle für alle Unternehmensbereiche dar, die bei Prozess- und Finanzanalysen als Datenbasis für geschäftliche Entscheidungsträger unabdingbar ist.

Im Gegensatz zu abteilungsbezogenen Systemstrukturen, i.e. Insellösungen, ist es durch den Einsatz eines ERP-Systems möglich Funktionen zu nutzen, für deren Durchführung Informationen aus mehreren Abteilungen nötig sind [7]. So kann zum Beispiel bei Eingang eines Auftrags sofort automatisiert anhand verschiedenster Kriterien 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 [7]. 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.

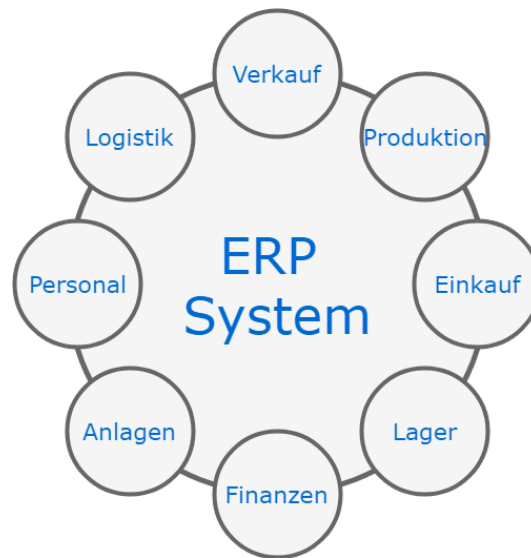


Abbildung 2.1: Schematische Darstellung: Typische Modulgliederung von ERP-Systemen.

Um die umfangreichen Funktionalitäten eines ERP-Systems zu gliedern und aufzuteilen, bedienen sich die meisten Hersteller eines Modul-Systems [13]. Meist spiegelt die Aufteilung dieser Module die einzelnen Abteilungen eines Unternehmens wider. 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 nach und nach in Betrieb genommen werden. Zu den bekanntesten ERP Herstellern zählen unter anderen IBM, SAP, Microsoft, Infor und Sage [8][3].

2.2 Geschichte und Grundarchitektur Dynamics 365 Business Central

2.2.1 Geschichte

Das ERP-System, das heute *Microsoft Dynamics 365 Business Central* heißt, erschien ursprünglich 1984 unter dem Namen *PCPlus* als ein ERP-System für Microsoft DOS in Dänemark [6]. Während seiner mittlerweile 35-jährigen Geschichte wurde das Produkt einige Male neu benannt und an den technischen Fortschritt der Industrie angepasst. Was 1984 begann, wurde 1995 unter dem Namen *Navision Financials* als das erste ERP-Produkt mit grafischer Benutzeroberfläche für Windows95 präsentiert. Im Jahr 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.

2.2.2 3-Schichten Architektur

Microsoft 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 Business Central unterscheiden wir zwischen Endbenutzerschicht, Serverschicht und der Datenbankschicht [18][6].

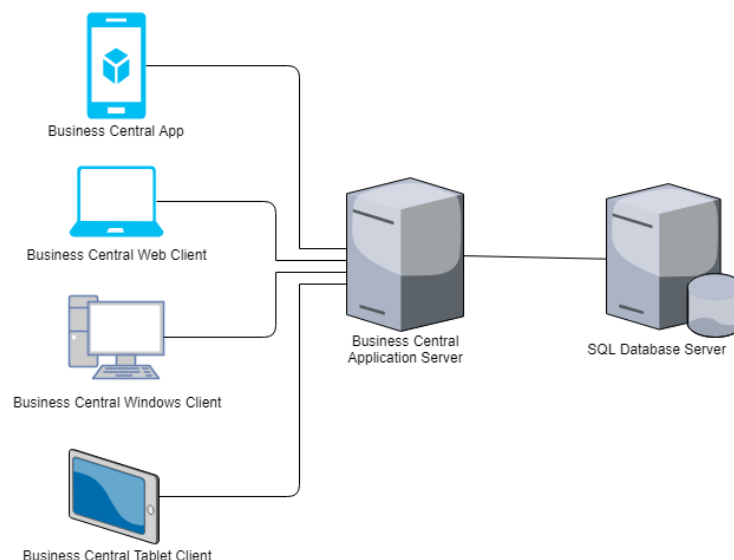


Abbildung 2.2: Dynamics 365 Business Central: 3-Schichten Architektur

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 Web Client und die Business Central Mobile App. Aber auch von Drittanbietern erstellte Softwarekomponenten, die die Microsoft Graph API oder von Business Central veröffentlichte Webdienste nutzen, sind der Endbenutzerschicht zuzuordnen.

Schicht 2: Die Serverschicht

Die Business Central Serverapplikation, auch *Middle-Tier* oder *Service-Tier* genannt, stellt das Herzstück des Gesamtsystems dar [10]. 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 Web Clients. Die Webserverkomponente selbst ist eine ASP.NET Applikation, die auf einem mitgeliefertem IIS (Internet Information Server) Web 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. Aufgrund 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 so zu Systemausfällen führen können.

2.3 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 [17]. Diese *Objekte* sind nicht mit jenen aus der klassischen objektorientierten Programmierung zu vergleichen. Dynamics 365 ist objektbasiert, jedoch 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	✓	✓
TableExtension	✗	✓
Page	✓	✓
PageExtension	✗	✓
Report	✓	✓
Codeunit	✓	✓
Query	✓	✓
XMLPort	✓	✓
MenuSuite	✓	✗
Enum	✗	✓

Tabelle 2.1: 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 gespeichert werden. 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 auf die beschriebenen Daten zu definieren.

TableExtension

Tabellenerweiterungen sind ein zentraler Baustein der Erweiterungsentwicklung mit AL. Sollten an einem bestehenden Tabellenobjekt Änderungen oder Erweiterungen nötig sein, würde man in C/AL einfach das bestehende Tabellenobjekt abändern. Mit dem Konzept der Erweiterungsentwicklung 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 zu-

sätzlich 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.

dbo.CRONUS AT\$Cust_ Ledger Entry PK Entry No_ Customer No_ Posting Date	dbo.CRONUS AT\$Cust_ Ledger Entry\$3d5b2137-efeb-4014-8489-41d37f8d4c3 PK Entry No_ ExtensionField1 ExtensionField2
--	---

Abbildung 2.3: Dynamics 365 Business Central: Standardtabelle und dazugehörige Tabellenerweiterung.

Page

Seiten (engl. "Pages") 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. Die Eigenschaften einer Seite 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 (engl. "Page Extensions") 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 (engl. "Reports") 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

Quelltexteinheiten (engl. "Codeunits") stellen die Logikkomponenten des Systems dar. Sämtliche Berechnungen, Buchungsroutinen und andere Teile der Geschäftslogik werden hier abgebildet. Um die Funktionalität einer Geschäftslogik abzuändern, ist es unter C/AL üblich direkt den verantwortlichen Code dafür abzuändern. Unter AL können

bestehende Codeunits des Microsoft-Standards nicht geändert werden. Um unter AL Änderungen durchzuführen, müssen Event-Subscriber erstellt werden, die sich in die Standardlogik einklinken.

Query

Abfrageobjekte (engl. "Queries") 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

Um strukturierte Daten einfach in das Programmsystem zu importieren und aus dem System zu exportieren werden Objekte vom Typ XMLPort verwendet. Trotz des Namens, sind XMLPorts nicht nur auf das XML Format beschränkt, sondern können auch verwendet werden um CSV und andere Formate zu verarbeiten.

MenuSuite

Menüobjekte (engl. "MenuSuites") bestimmen die hierarchische 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

Enums definieren Aufzählungstypen und ersetzen den in C/AL verwendeten *Option* Datentyp, der lediglich eine kommaseparierte Zeichenfolge darstellt, schlecht erweiterbar ist, und somit für die Erweiterungsentwicklung unter AL nicht zielführend ist. Enumerationen sind nur unter AL nutzbar.

Kapitel 3

Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung

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.

3.1 Aufgabenstellung

Für die Kunden des fiktiven Auftraggebers der betrachteten 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 Webdienst 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.

3.2 Entwicklungsprozess

Folgend werden die Unterschiede im Entwicklungsprozess zwischen den beiden Programmierparadigmen aufgezeigt. Dies beginnt bei den verwendeten Entwicklungsumgebungen und zieht sich durch den gesamten Entwicklungsprozess der einzelnen Objektarten, welche die schlussendliche Erweiterung darstellen.

3.2.1 Entwicklungsumgebung

Entwicklungsumgebungen stellen das Hauptwerkzeug des Entwicklers dar [14]. Aufgabe einer Entwicklungsumgebung ist es, dem bedienenden Entwickler eine umfassende und komfortabel bedienbare Reihe von Funktionen und Werkzeugen zu bieten, um schlussendlich möglichst effizient und fehlerfrei Quelltext zu verfassen.

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, seitdem 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 konstante Verbindung zum Applikationsserver wird gebraucht um Änderungen kompilieren und in weiterer Folge ausführen zu können.

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

3. Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung 14

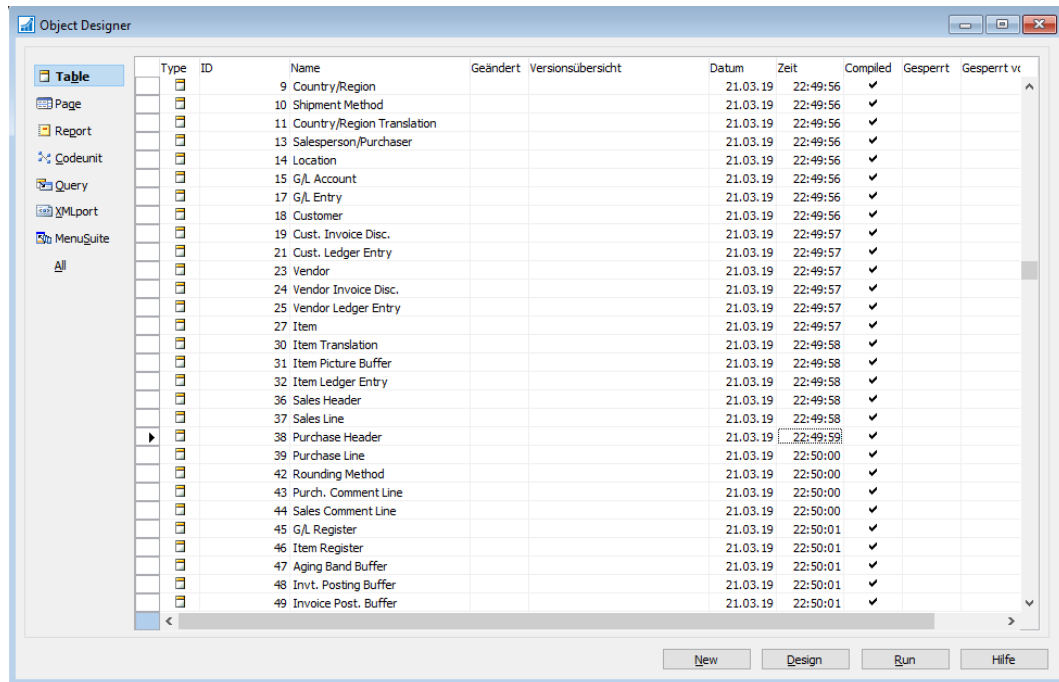


Abbildung 3.1: Development Environment: Übersicht über sämtliche Objekte und Objektarten mit dem Object Designer.

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. Mithilfe des Designers gelangt man ebenfalls zum C/AL Editor, in dem die Implementierung der Geschäftslogik erfolgen kann.

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.

3. Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung 15

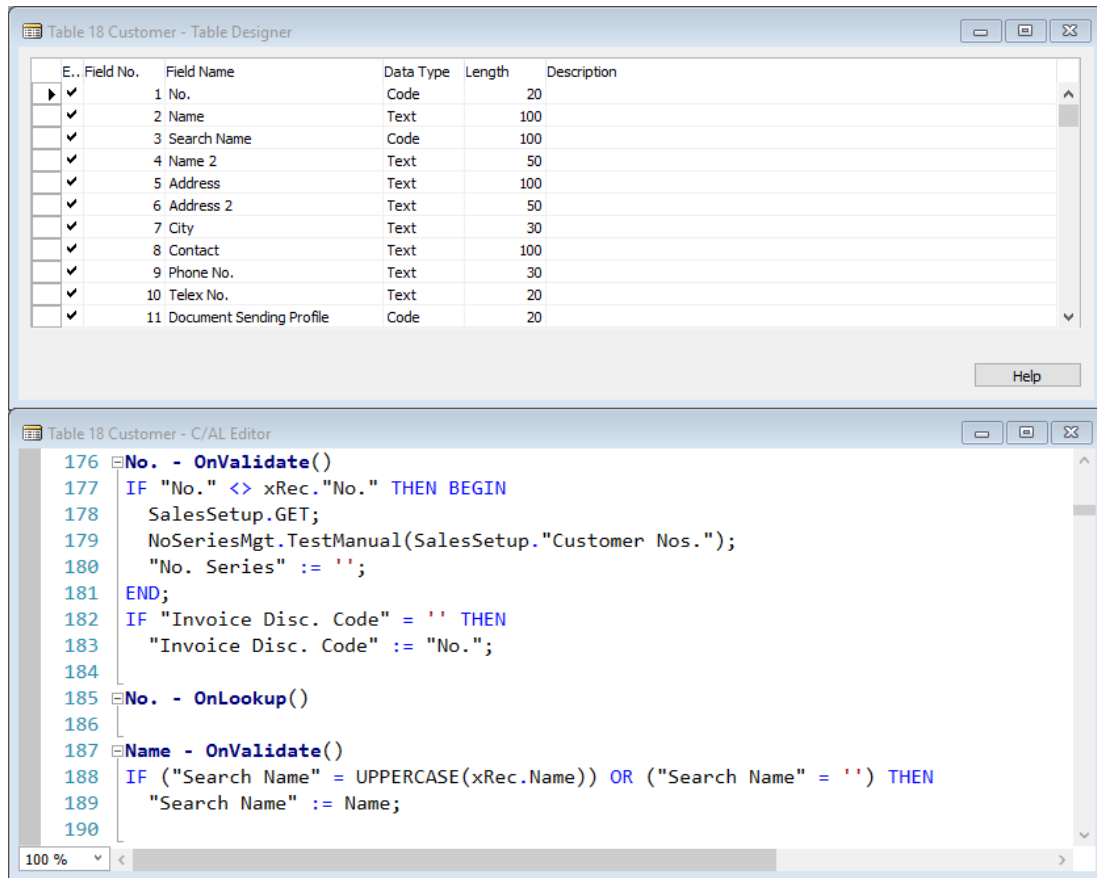


Abbildung 3.2: Development Environment: Table Designer und C/AL Code Editor der Debitoren Tabelle.

Um Applikationsobjekte zwischen verschiedenen Datenbanken zu transferieren, und so 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 mit der Dateiendung .txt ist zwar grundsätzlich durch den Menschen lesbar, definiert jedoch auch einige intern nötige Eigenschaften. Andererseits können Objekte auch im Binärformat exportiert werden, erkennbar durch die 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.

3. Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung 16

AL - Visual Studio Code

Die Programmierung von Erweiterungen für Microsoft Dynamics 365 Business Central erfolgt in Visual Studio Code [14]. Visual Studio Code ist ein OpenSource Quelltext-Editor für verschiedenste Programmier- und Markup Sprachen basierend auf dem Electron ¹ Framework. Visual Studio Code ist in der Sprache Typescript implementiert, und setzt im Gegensatz zum Development Environment kein Windows-Betriebssystem voraus, sondern kann auch unter Mac und Linux verwendet werden. Als zeitgemäße Entwicklungsumgebung liefert Visual Studio Code eine Auswahl einiger Funktionalitäten 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

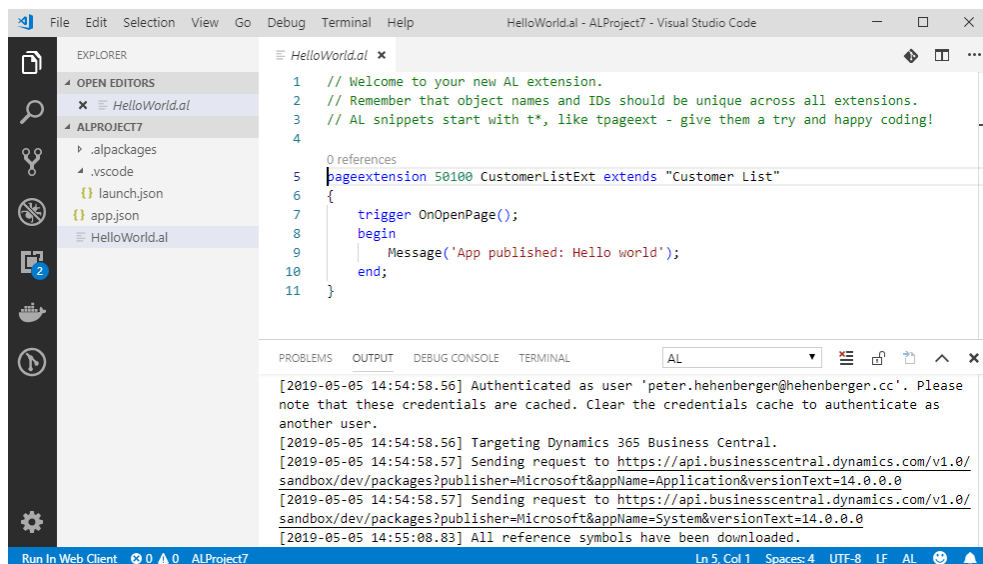


Abbildung 3.3: Visual Studio Code: Grafische Oberfläche, Beispielhafte AL - Erweiterung

¹<https://electronjs.org/>

3. Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung 17

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 installiert werden musste, und je Version des Systems eine eigenständige Installation von Nöten ist, um auch vorangegangene Versionen zu unterstützen.

Visual Studio Code in Kombination mit AL ist rein textbasiert. Die aus dem Development Environment bekannten 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 Entwicklertaschine. 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 enthalten die 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. Dadurch lässt sich der geschriebene Quelltext sinnvoll und ohne Umwege in einem Source Code Management System wie Git verwalten.

3.2.2 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.

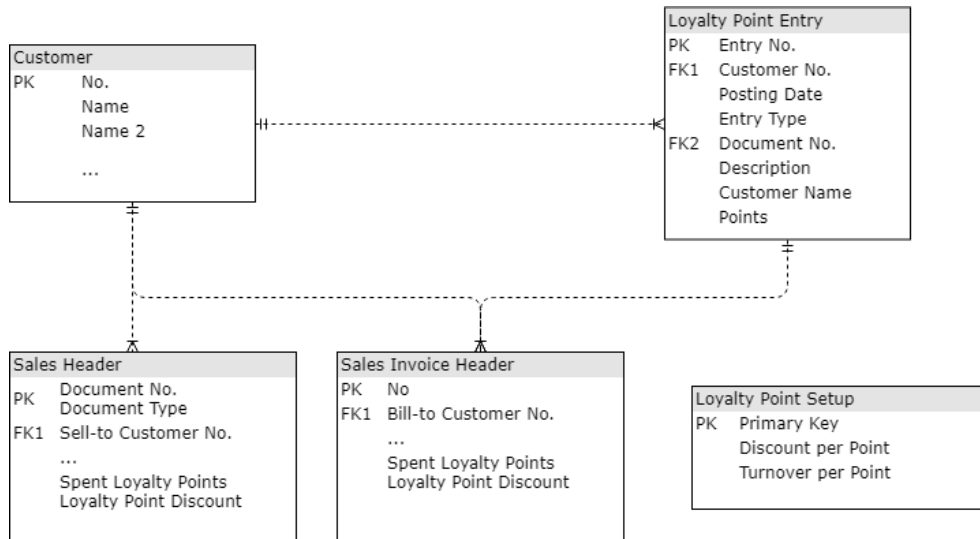


Abbildung 3.4: Grundfunktionalität: Tabellenschema der Erweiterung zur Treuepunktverwaltung.

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 betroffenen 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 Setup* 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 per 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

3. Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung 19

auch genug verfügbare Treuepunkte hat, um den Rechnungsrabatt mit seinem Punktekonto 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 Applikationsobjekt erstellt, deren Prozeduren aus allen anderen Applikationsobjekten aufgerufen werden können.

Implementierung der Grundfunktionalität in C/AL

Die folgende Implementierung setzt auf eine bereits existierende Standardumgebung von *Microsoft Dynamics 365 Business Central* auf. Die nötigen beschriebenen Tabellenänderungen und neuen Tabellen können unter C/AL im Development Environment fast ausschließlich mithilfe des *Table-Designer* Fensters erstellt werden.

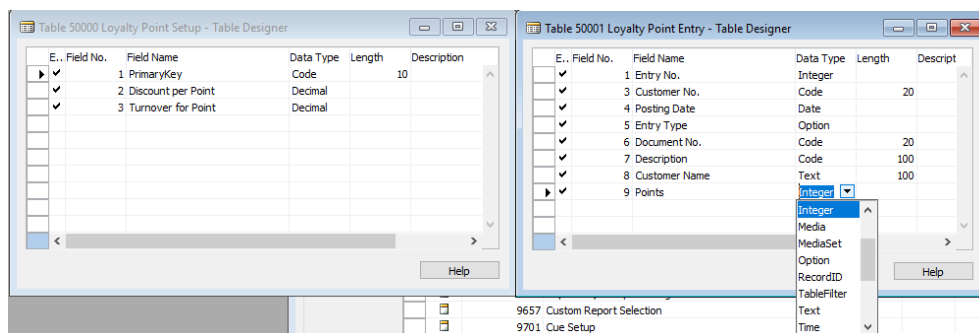


Abbildung 3.5: Grundfunktionalität: Tabellen der Treuepunkterweiterung.

Beim Speichern der erstellten Tabellenobjekte werden diese vom Server direkt kompiliert und anschließend am Datenbankserver das Datenbankschema entsprechend erweitert. Neben den beiden in Abbildung 3.5 erstellten Tabellen, sind auch Änderungen der Tabelle *Sales Header* und *Sales Invoice Header* nötig, siehe Abbildung 3.4. 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 vorhandenen 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 3.1: Validierung Treuepunkteinlösung.

```

1 Trigger "Spent Loyalty Points" - OnValidate()
2
3 // Make sure 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 just 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* [16]. 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 im Programmstück 3.1 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 definierten 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.

Implementierung der Grundfunktionalität in AL

Im Gegensatz zu C/AL werden unter AL Änderungen nicht direkt im bestehenden Standardsystem vorgenommen, sondern werden in abgekoppelten Erweiterungen zusammengefasst. Um eine Erweiterung zu erstellen und verwalten zu können, sind einige Metadaten nötig. Diese Metadaten sind im JSON Format in Form einer Datei namens `app.json` der Erweiterung beizufügen.

Programm 3.2: 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
8     spend Loyalty Points, as well as Exports and Reporting capabilities to serve BI
9     needs",
10  "dependencies": [],
11  "screenshots": [],
12  "platform": "14.0.0.0",
13  "application": "14.0.0.0",
14  "idRanges": [
15    {
16      "from": 50200,
17      "to": 50249
18    }
19  ],
20  "contextSensitiveHelpUrl": "https://localhost/LoyaltyPoints/",
21  "showMyCode": true,
22  "runtime": "3.0"
23 }
```

Neben den gezeigten Eigenschaften stehen in der `app.json` noch einige andere Eigenschaften 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 des fiktiven 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

3. Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung 22

zu vergeben. Nach der Eingabe der entsprechenden Informationen soll nach Bestätigung durch den Benutzer der Prozess zur Vergabe der Treuepunkte anhand der konfigurierbaren 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 Berichtobjekten 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 Berichtobjekts 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 Articleinstandspreise 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 Berichtobjekt 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:

EDIT - LP GRANT LOYALTY POINTS

Options

Points to Grant 50

Document No. MARKETING-04/2019

Description Loyalty point grant 04/2019

Customer

Show results:

Where: No. ▼ is: 10000..D00010 ▼

And: Post Code ▼ is: >=4000&<5000 ▼

And: LP Loyalty Points ▼ is: <5000

OK Cancel

Abbildung 3.6: Grundfunktionalität: Business Central Web Client Request Page, Parametrisierung und Filterkriterien zur Treuepunktvergabe.

3.2.3 Datenträgerexport

Datenträgerexports werden unter Microsoft Dynamics 365 Business Central vorwiegend mithilfe der XMLPort-Applikationsobjektart durchgeführt. Diese Objektart steht sowohl in AL, als auch in C/AL zur Verfügung. Bei der Nutzung und Verwendung ergeben sich jedoch einige Unterschiede.

Implementierung des Datenträgerexports in C/AL

. 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.

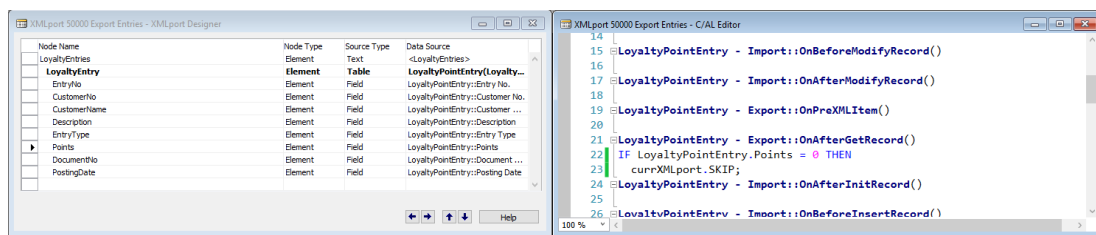


Abbildung 3.7: Grundfunktionalität: Datenexport, XMLPort Designer und C/AL Editor.

Innerhalb der definierten Trigger steht es dem Entwickler frei, Code zu platzieren. Innerhalb der Trigger 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 Treuepunktposten stellt keine Anforderungen an ein komplexes Datenformat, hier genügt einzig die Definition der zu exportierenden Datensätze im XMLPort-Designer Fenster um an das gewünschte Ziel zu kommen.

Implementierung des Datenträgerexports in 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 schwerwiegendste Unterschied zwischen 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 aufgrund der technischen Einschränkungen nicht möglich. In der serverlosen Variante wird rein Anhand von Datenströmen (*Streams*) gearbeitet, die im Hauptspeicher der Servermaschine verwaltet werden. Das umgeht 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 gespeicherten zu verarbeitenden Daten. Aufgaben dieser Art werden so schnell komplex, machen einige Überlegungen notwendig, und enden meist darin, dass ein Cloud-Datenspeicher angeschafft 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.

3.2.4 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 realisierte 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 System *Microsoft PowerBI* zu übermitteln.

Aber auch innerhalb des Systems finden sich verschiedene Möglichkeiten, Auswertungen zu gestalten. Zum einen lassen sich *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 im Unternehmen entsprechende Diagramme zu erstellen, um ihn bei seiner täglichen Arbeit zu unterstützen.

Die meistgenutzte Möglichkeit für grafische Auswertungen und Datenaufstellungen 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 ein normiertes XML-Dokument. Dieses Dataset wird im zweiten Schritt in ein Layout übertragen, das 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.

3.2.5 Webservice-Anbindung

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

Implementierung der Webservice-Anbindung in C/AL

Die in der Sprache C/AL 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 3.3: .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("
10 application/json"));
11             HttpContent content = new StringContent(dto.ToJsonString(), Encoding.UTF8, "
12 application/json");
13             HttpResponseMessage response = client.PostAsync(endpoint, content).Result;
14             return (int)response.StatusCode;
15         }
16         catch (Exception e)
17         {
18             return 0;
19         }
20     }
21 }

```

Die Methode *TransmitEntry* im Programmstück 3.3 empfängt hier ein Objekt vom Typ *LoyaltyPointEntryDTO*, das 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 ungewollte Fehler auftreten wenn Nullable-Typen verwendet werden. In C/AL wird das Konzept von Null nicht unterstützt, und der Versuch Null zurückzuliefern führt zu einem Laufzeitfehler. Es empfiehlt sich daher, wenn möglich nur Ganzzahlen und Zeichenfolgen als Rückgabewerte zu verwenden. Anstatt von Null Werten müssen Standardwerte definiert werden. 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 empfiehlt sich daher Ausnahmen dort zu

3. Vergleich der erweiterungsbasierten Entwicklung zur konventionellen Codeanpassung 28

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.

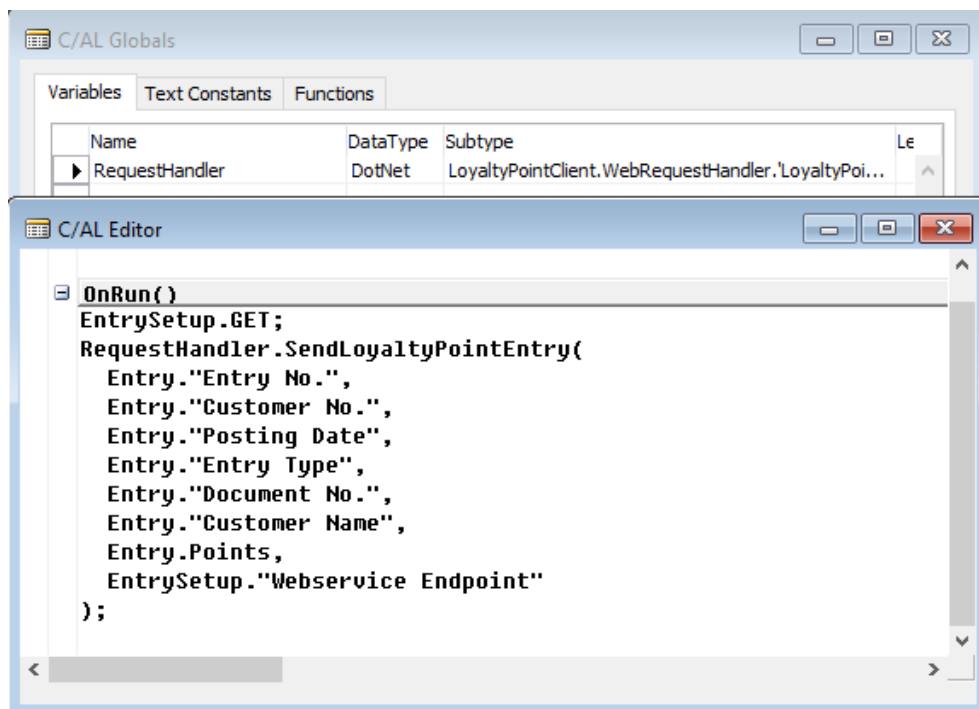


Abbildung 3.8: Development Environment: Nutzung eines DotNet Datentypen zur Kommunikation mit einem CRM-Webdienst.

Implementierung der Webservice-Anbindung in 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 Funktionalitäten 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 um die neuen Komponenten erweiterte Sprache AL den 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, siehe Programm 3.4, erstellt werden, die beliebige *Http*-Webanfragen erstellen und absenden kann. Die Funktion *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 definierte *Http*-Anfrage absendet, und die Antwort in Form eines Ausgabeparameters *Response* bereitstellt.

Programm 3.4: AL: Hilfsprozedur zur Übermittlung von POST-Anfragen an Webdienste.

```

1 procedure DoPostRequest(Endpoint: Text;
2                         RequestHeaders: Dictionary of [Text, Text];
3                         ContentHeaders: Dictionary of [Text, Text];
4                         Body: Text;
5                         var Response: HttpResponseMessage)
6 var
7   Client: HttpClient;
8   Request: HttpRequestMessage;
9   Headers: HttpHeaders;
10  Content: HttpContent;
11  DictKey: Text;
12 begin
13   Request.Method := 'POST';
14   Request.SetRequestUri(Endpoint);
15   Request.GetHeaders(Headers);
16
17   foreach DictKey in RequestHeaders.Keys() do
18     Headers.Add(DictKey, RequestHeaders.Get(DictKey));
19
20   Content := Request.Content();
21   Content.WriteFrom(Body);
22
23   Content.GetHeaders(Headers);
24   foreach DictKey in ContentHeaders.Keys() do begin
25     Headers.Remove(DictKey);
26     Headers.Add(DictKey, ContentHeaders.Get(DictKey));
27   end;
28   Request.Content(Content);
29   Client.Send(Request, Response);
30 end;
31

```

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 365 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. Die Prozedur `SendRequests()`, siehe Programm 3.4, empfängt eine vorgefilterte Record-Variable von Treuepunktposten, und erstellt für jeden Posten ein neues JSON-Dokument *EntryObject*, welches in weiterer Folge mithilfe der Hilfsprozedur `DoPostRequest()`, siehe Programm 3.3, an den Webdienst übermittelt wird.

Programm 3.5: 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 4

Evaluierung & Diskussion

4.1 .NET Klassenbibliothek und AL-Sprachkonstrukte

4.1.1 Testaufbau

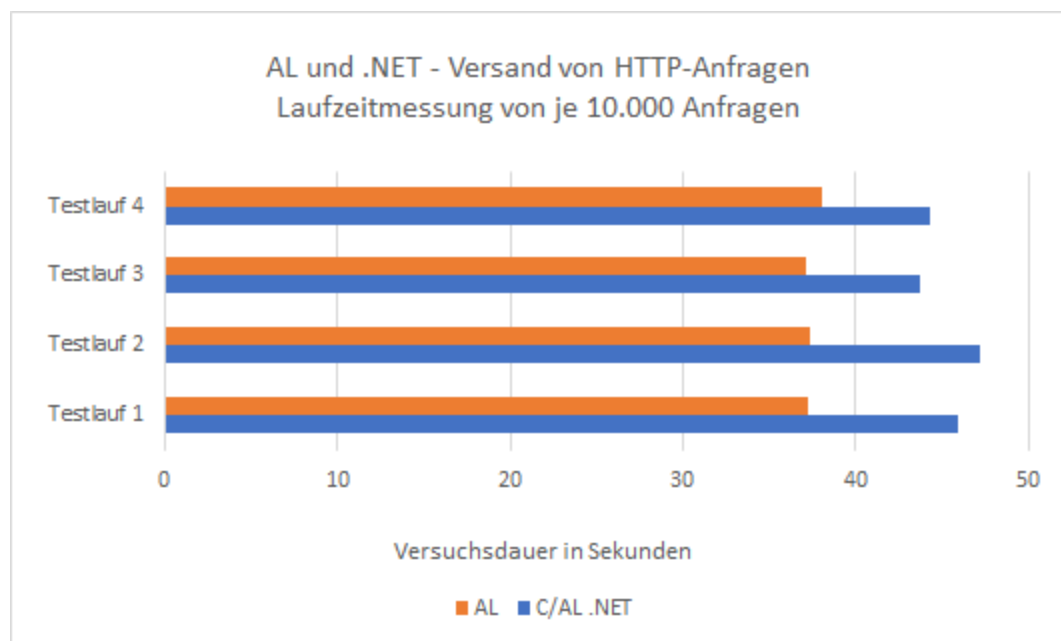
Im ersten Testaufbau werden die im Treuepunkt-Beispiel 3.1 verwendeten Varianten der Webdienst Kommunikation gemessen und auf ihr Laufzeitverhalten untersucht. In diesem Test werden beide Varianten eine große Anzahl von Http-Anfragen erstellen und an einen Webdienst übermitteln. Um die Ergebnisse vergleichbar 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 implementierte Webdienst und die ausführende Microsoft Dynamics 365 Business Central Instanz laufen auf dem selben Rechner. 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 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.

4.1.2 Messergebnisse: Erstellung von Http-Anfragen

Tabelle 4.1: Ergebnisse für 10000 Anfragen, Messwerte in Sekunden.

	Testlauf 1	Testlauf 2	Testlauf 3	Testlauf 4	Durchschnitt	Std. Abweichung
C/AL .NET	45,960	47,174	43,707	44,280	45,280	1,371
AL	37,247	37,396	37,166	38,016	38,016	0,334
Differenz	8,713	9,778	6,541	6,264	7,824	1,038

**Abbildung 4.1:** 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 etwa 20% performanter.

4.2 Tabellenanpassung und Tabellenerweiterung

Wie im Unterkapitel *Objektarten in Business Central 2.3* 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 in 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 *SumField* 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 erstellten Feld zu quantifizieren. Zu diesem Zweck wird die C/AL Variante der Tabelle *Loyalty Point Entry* 3.4 mittels einer neuen Erweiterung um ein neues Feld *Points2* erweitert, sodass sich folgendes Tabellenschema ergibt:

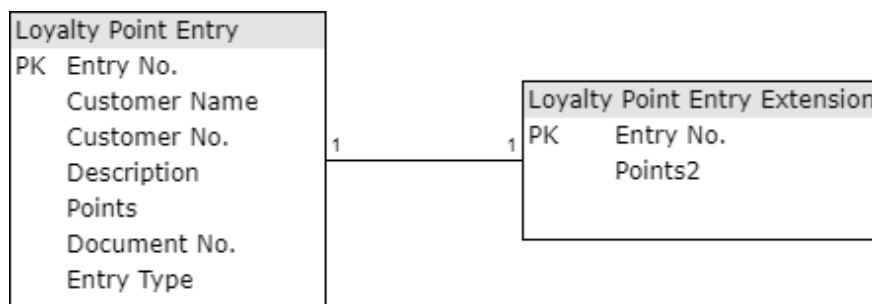


Abbildung 4.2: Tabellenschema Treuepunktposten und zugehörige Erweiterung.

Die Abfrage um herauszufinden, wie viele Treuepunkte ein Kunde momentan hat, soll möglichst effizient gestaltet werden. Hierfür wird in der C/AL Variante - also für das 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 bestehend aus Datenbank und Serverinstanz nach jeder Messung neu gestartet, um Effekte durch Zwischenspeicherungen auszunehmen.

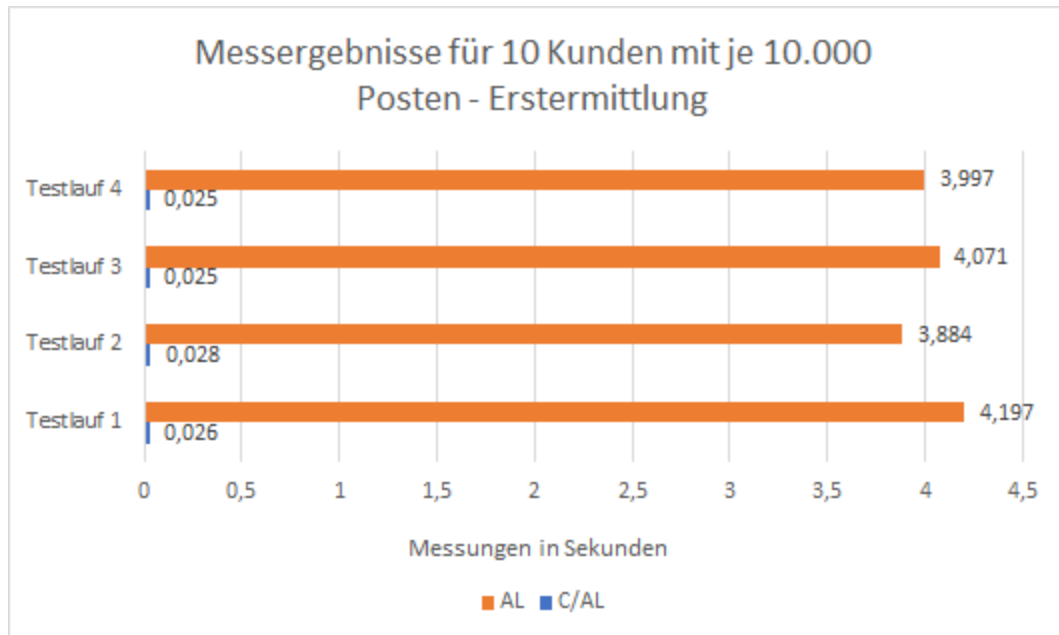


Abbildung 4.3: Messung: Aggregation von Treuepunktposten - Ohne Zwischenspeicher.

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 [12]. 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.

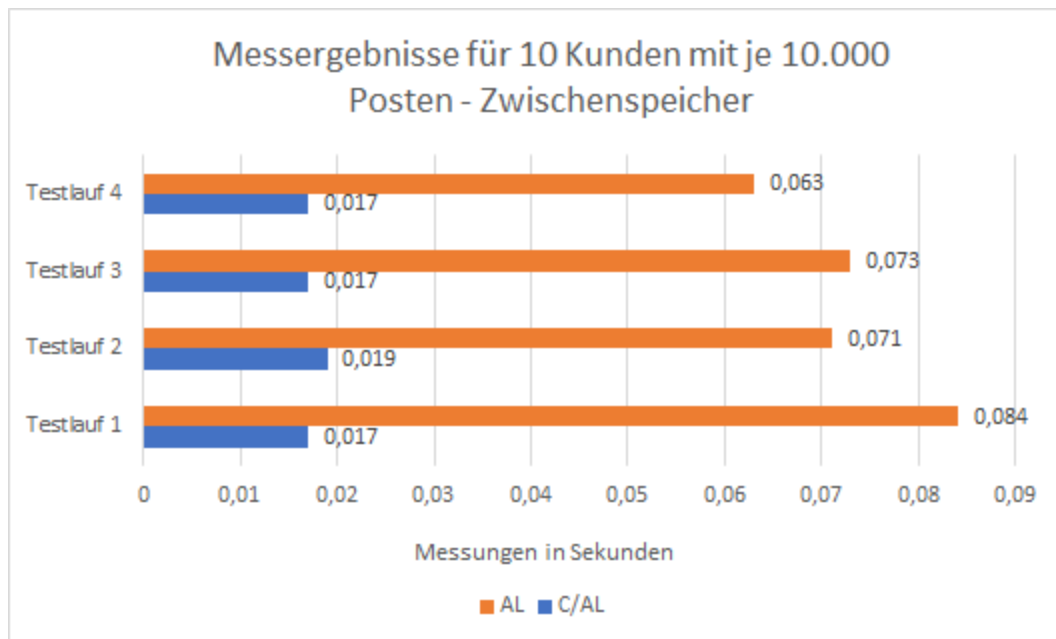


Abbildung 4.4: Messung: Aggregation von Treuepunktposten - mit Zwischenspeicher.

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

4.3 Codeanpassung und ereignisorientierte Codeerweiterung

Mit der Einführung der Erweiterungsentwicklung und AL, werden den Entwicklern 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 er-

möglichen, 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 gesamten 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;
```

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 der Ereignismechanismus auch seine Einschränkungen und Schwächen:

Ausführungsreihenfolge

Was passiert, wenn mehrere Programmstücke auf dasselbe Ereignis reagieren wollen? Konkret geht es darum um die Ausführungsreihenfolge. Welches der reagierenden Codestücke wird 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.

Zugriffseinschränkung

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 Basisapplikation 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.

4.4 Source Code Management 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 [15]. Programmcode wird nicht mehr primär in einer Entwicklungsdatenbank bearbeitet und gespeichert, sondern befindet sich lokal auf dem Rechner 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 Entwicklerkonferenz des Jahres für Dynamics 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 gaben an für C/AL Programmcode ein entsprechendes Verwaltungssystem zu verwenden. 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 daher einen zusätzlichen Mehraufwand dar, und verkompliziert dadurch den Entwicklungsprozess. Hier ist es oft einfacher und zielführender, nächtlich automatisiert Sicherungen der Entwicklungsdatenbank beziehungsweise aller Applikationsobjekte zu erstellen, und diese bei Bedarf wiederherzustellen.

Mit AL und Visual Studio Code sind diese Einschränkungen Vergangenheit. Als Entwickler für Microsoft Dynamics 365 Business Central hat man nun Zugang zu einer modernen Entwicklungsumgebung mit vollständiger Integration in die Quellcode-Verwaltungssoftware Git.

Mit all diesen Neuerungen werden nun auch die Themen *Continuous Integration* und *Continuous Deployment* - kurz *CI/CD* für Business Central Entwickler immer interessanter. Bei *Continuous Integration* handelt es sich um ein Vorgehen in der Softwareentwicklung, bei dem Mitarbeiter an einem Softwareprojekt die von ihm getätigten Änderungen mehrmals täglich in den Hauptentwicklungsstrang integriert [5]. Die durchgeführten Änderungen werden im Laufe dieser Integration sofort kompiliert und anhand verschiedener Kriterien getestet. Das wichtigste Testkriterium sind hier durch die Entwickler erstellten Testfälle. Sollte ein Testfall fehlschlagen, so wird die getätigte Änderung verworfen, und der verantwortliche Mitarbeiter darüber informiert. So kann schnellstmöglich auf Fehler reagiert werden.

Die Aufgabe des *Continuous Deployment* besteht darin, erfolgreich integrierte Änderungen auch direkt in Anwendungsform verfügbar zu machen. Im Business Central

¹www.mibuso.com

Umfeld bedeutet das typischerweise, automatisiert ein Online-Testsystem zu erstellen, das mit der aktuellen Version der entwickelten Erweiterung bestückt wird und so für Entwickler, Tester und auch den Endkunden zugänglich ist. Das erstellte Testsystem arbeitet vorzugsweise auf einer Kopie der Kundendatenbank, um möglichst realitätsnahe Verhältnisse zu schaffen.

Diese beiden Entwicklungsmodelle abzubilden ist mit dem Development Environment und AL schier unmöglich [20]. Einerseits gibt es keine Möglichkeit, Applikationsobjekte ohne der Nutzung des Development Environment zu kompilieren. Andererseits ist auch die Integration von bereits erstellten Applikationsobjekten im Binär- oder Textformat komplex und dadurch fehleranfällig. Da ein großer Teil der Entwickler C/AL Applikationen ohnehin auf zentralen Entwicklungsdatenbanken und ohne Quellcodeverwaltung entwickeln, stellt sich die Frage der Notwendigkeit eines *CI/CD* Systems oft erst gar nicht.

In der cloudbasierten Zukunft von Microsoft Dynamics 365 Business Central bringt der Einsatz derartiger *CI/CD* Systeme einige Vorteile. So können einzelnen Kunden ohne Mehraufwand aktuelle Testsysteme zur Verfügung gestellt werden, anhand deren neue Funktionalitäten präsentiert und getestet werden können. Aufgaben wie die Erstellung eines neuen Testsystems anhand der aktuellsten Versionen des Grundsystems mit der Eigenentwicklung, basierend auf Kundendaten können in der C/AL Welt durchaus mehrere Stunden in Anspruch nehmen. Mit einem cloudbasierten System und einem geeignetem *CI/CD* Prozess kann diese Aufgabe automatisiert, vorhersehbar und zeitnah innerhalb weniger Minuten ausgeführt werden. Leider gibt es seitens Microsoft bezüglich *CI/CD* noch keine offiziellen Empfehlungen oder bewährte Verfahren. So sind Entwickler in diesem Aspekt leider noch vollkommen auf sich selbst angewiesen. Es ist jedoch davon auszugehen, dass Microsoft hier in Zukunft entsprechende Konfigurationen zur Verfügung stellen wird.

Kapitel 5

Zusammenfassung & Ausblick

5.1 Zusammenfassung

Im Laufe dieser Arbeit wurde eine voll funktionale Erweiterung zur Verwaltung von Treuepunkten für Stammkunden entwickelt. Dafür wurden vorerst die verschiedenen Arten von Applikationsobjekten beleuchtet. Im weiteren Verlauf der Arbeit wurde die Erweiterung sowohl durch Codeanpassung in C/AL, als auch erweiterungsbasiert mit der neuen Sprache AL umgesetzt. Aus den Tests geht hervor, dass beide Varianten wie erwartet das gleiche Verhalten aufweisen. Auch die Laufzeitmessungen der beiden Varianten sind bis auf die im Kapitel Evaluierung & Diskussion hervorgehobenen architekturbedingten Sonderfälle durchaus auf dem selben Niveau. Mit dem erweiterten Funktionsumfang von AL im Vergleich zu C/AL lassen sich nun viele Probleme, die unter C/AL nur mit externen Klassenbibliotheken lösbar waren, nun direkt mithilfe der neuen Sprachkonstrukte lösen. Das größte Problem mit AL ist derzeit noch die Abhängigkeit auf von Microsoft veröffentlichten Ereignissen. Nichts desto trotz überwiegen die vielen Vorteile, angefangen von automatisierbaren Updates und einer zeitgerechten Entwicklungsumgebung, bis hin zur Möglichkeit bisher langwierige Aufgaben wie die Erstellung von online fähigen Testsystemen zu automatisieren.

5.2 Ausblick

Die in dieser Arbeit verwendeten Hauptversion 14 des Systems ist die letzte, in der Code Anpassungen in C/AL noch möglich sind. Mit der kommenden Hauptversion im Oktober 2019 werden Anpassungen des Systems nur noch via in AL entwickelten Erweiterung unterstützt [19]. Um künftige Iterationen des Systems zu unterstützen, müssen Entwickler somit ihre C/AL Anpassungen in AL Erweiterungen übersetzen. Traditionell sind die Partnerunternehmen bei der Umsetzung grober technologischer Änderungen jedoch sehr träge. Selbst die vergleichsweise kleine Umstellung auf die 3-Schichten-Architektur im Jahr 2009 beschäftigte Entwickler oft mehrere Jahre. Dementsprechend ist auch bei dieser nun größeren Umstellung, inklusive dem Sprung in die Cloud damit zu rechnen, dass der Umstieg einige Jahre in Anspruch nehmen wird. Denn Partnerunternehmen müssen natürlich weiterhin ihre Bestandskunden mit älteren Versionen unterstützen, und sollen nebenher Neuprojekte auf Cloudbasis mit einer neuen Programmierspra-

che und einem stark abgewandelten Entwicklungskonzept umsetzen. Dennoch ist davon auszugehen, dass sich die Cloudvariante von Microsoft Dynamics 365 Business Central zumindest mittelfristig durchsetzen wird, und es somit für Entwickler an der Zeit ist, sich der vertrauten C/AL Welt ein Stück weit abzuwenden, und den Sprung in das erweiterungsbasierte Konzept von AL zu wagen.

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> (besucht am 25.05.2019) (siehe S. 3).
- [2] David Tein Bernard Wong. „Critical Success Factors for ERP Projects“ (Jan. 2003), S. 1–2. URL: https://www.researchgate.net/publication/229022123_Critical_Success_Factors_for_ERP_Projects (besucht am 25.05.2019) (siehe S. 3).
- [3] Bernhard Gärtner Birgit Feldbauer-Durstmüller. „Controlling und ERP-Systeme: Aktuelle Ergebnisse der eingesetzten ERP-Systeme im Controlling in Österreich“. *Controller Magazin* (Juli 2013) (siehe S. 6).
- [4] 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> (besucht am 25.05.2019) (siehe S. 4).
- [5] Martin Fowler und Matthew Foemmel. „Continuous integration“. *Thought-Works* <http://www.thoughtworks.com/ContinuousIntegration.pdf> 122 (2006), S. 14 (siehe S. 38).
- [6] Michaela Gayer. *Microsoft Dynamics NAV - Einführung in Design und Programmierung*. mbst books, 2016 (siehe S. 7).
- [7] Michaela Gayer, Christian Hauptmann und Jürgen Ebert. *Microsoft Dynamics NAV 2018: Das Anwenderbuch zur Abwicklung von Geschäftsprozessen*. Carl Hanser Verlag GmbH Co KG, 2018 (siehe S. 5).
- [8] Md Hoque, Dania Bahssas und Adnan Albar. „Enterprise Resource Planning (ERP) Systems: Design, Trends and Deployment“. *The International Technology Management Review*, Vol. 5 (2015), (Juni 2015), S. 72–81 (siehe S. 6).
- [14] Tobias Kahlert und Kay Giza. *Visual Studio Code Tips and Tricks*. Microsoft Press, 2016. URL: <https://www.microsoft.com/germany/techwiese/aktionen/visual-studio-code-ebook-download.aspx> (besucht am 25.05.2019) (siehe S. 13, 16).

- [9] 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> (besucht am 25.05.2019) (siehe S. 5).
- [10] David Roys und Vjekoslav Babic. *Implementing Microsoft Dynamics NAV 2009*. Packt Publishing Ltd, 2009 (siehe S. 8).
- [11] Simona Sternad und Samo Bobek. „Success Factors for Dynamics NAV Solution Implementation : What Matters and How“. In: 2012 (siehe S. 3).
- [12] Jörg A. Stryk. *The NAV/SQL Performance Field Guide*. Books on Demand, 2014 (siehe S. 35).
- [13] David Studebaker. *Programming Microsoft Dynamics NAV*. Packt Publishing Ltd, 2007 (siehe S. 6).

Online-Quellen

- [15] Freddy Kristiansen. *Developing Business Central Extensions*. 2018. URL: <https://community.dynamics.com/business/b/freddysblog/archive/2018/11/12/developing-business-central-extensions-part-1-prerequisites> (besucht am 25.05.2019) (siehe S. 38).
- [16] Microsoft. *Business Central Developer Help*. 2019. URL: <https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/methods-auto/record/record-data-type> (besucht am 01.05.2019) (siehe S. 20).
- [17] Microsoft. *Designing Application Objects*. 2016. URL: <https://docs.microsoft.com/en-us/dynamics-nav/designing-application-objects#object-oriented-vs-object-based> (besucht am 01.05.2019) (siehe S. 9).
- [18] Microsoft. *Product and Architecture Overview*. 2017. URL: <https://docs.microsoft.com/en-us/dynamics-nav/product-and-architecture-overview> (besucht am 22.05.2019) (siehe S. 7).
- [19] Roberto Stefanetti. *Goodbay to C/SIDE and Windows Client!* 2019. URL: <https://robertostefanettinavblog.com/2019/04/01/goodbay-to-c-side-and-windows-client/> (besucht am 25.05.2019) (siehe S. 40).
- [20] Eric Wauters. *Developing Business Central Extensions/Apps in Team*. Juli 2018. URL: <http://www.waldo.be/2018/07/11/developing-business-central-extensionsapps-in-team/> (besucht am 25.05.2019) (siehe S. 39).