HOCHSCHULE FURTWANGEN UNIVERSITY HFU

Bachelorarbeit

in

Allgemeine Informatik

Konzeption und Implementierung einer Ablaufsteuerung für eine Low-Code-Plattform

Vorwort

Während meines Praxissemesters habe ich intensiv mit der Windows Workflow Foundation gearbeitet. Der Ansatz mittels einzelner und wiederverwendbarer Bausteine eine komplexe Funktion zusammenzubauen gefällt mir sehr. Doch leider hat sich während meiner Arbeit mit der Windows Workflow Foundation herausgestellt, dass man in bestimmten Szenarien an die Grenzen dieses Systems kommt. Dies liegt daran, dass die WWF nicht (wie im Sinne des Erstellers) zur Modellierung von langlaufenden Prozessen, sondern zur Implementierung einer Software verwendet wurde. Die WWF wurde als Low-Code-Plattform verwendet - wofür sie eigentlich nicht direkt konzipiert wurde. Features der WWF, wie das Persistieren nach jedem Zwischenschritt, werden z.B. bei einer LCP nicht benötigt und sorgen zudem für eine schlechte Performance. Die WWF hat auch keine Threadsicherheit und führt einen Ablauf meistens mit mehreren Threads abwechselnd aus. Zusätzlich ist der Einstieg in die WWF für einen Programmierer nicht intuitiv, da z.B. bestimmte Regeln über den Kontext der WWF Activities beachtet werden müssen.

Nach einer Recherche konnte ich keine im .NET Umfeld kostenlose Open Source Low-Code-Plattform bzw. Ablaufsteuerung finden, die meinen Anforderungen entspricht.

In dieser Abschlussarbeit möchte ich deshalb mein Konzept und eine Implementierung vorstellen, die als Alternative zur WWF verwendet werden kann, um damit Abläufe einer Software zu modellieren.

Abstract

This bachelor thesis discusses the conception and implementation of a flow control system which can be used in a low-code-platform. Using this system, individual software modules can be combined into a flow. This can be achieved with a graphical web-based editor using drag and drop. Information between these individual components were exchanged by parameters and local variables. It is also possible to add program code to the flow directly. The concept is implemented in C# .NET Core and can provide the fundamentals of a software across different platforms or can be integrated into existing software. In addition, other tools such as a debugger, flow validation and a flow repository were implemented to enhance the use of the software in practice.

Diese Bachelorarbeit beschäftigt sich mit der Konzeption und Implementierung einer Ablaufsteuerung für eine Low-Code-Plattform. Einzelne Software-Bausteine können, mit dem in dieser Arbeit erstellten System, in einem grafischen webbasierten Editor mittels Drag and Drop zu Abläufen zusammengesetzt werden. Mittels Parameter und lokale Variablen werden die Daten zwischen den einzelnen Bausteinen ausgetauscht. Zusätzlich kann ein Ablauf auch direkt mit Programmcode ergänzt werden. Das Konzept ist in C# mit .NET Core implementiert und kann plattformübergreifend die Grundlage einer Software bieten oder in eine andere, bereits vorhandene, Software integriert werden. Zusätzlich wurden weitere Werkzeuge wie z.B. ein Debugger, Validierung der Abläufe und ein Ablauf Repository implementiert, um den Einsatz der Software in der Praxis zu verbessern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BPM	Business Process Management
BPMN	Business Process Model and Notation
DAP	Debug Adapter Protocol
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
LCP	Low-Code-Plattform
RSS	Rich Site Summary
SaaS	Software as a Service
TCP	Transmission Control Protocol
WWF	Windows Workflow Foundation

1 Einleitung

Eine Low-Code-Plattform ist eine Entwicklungsumgebung, die es ermöglicht Software mithilfe eines visuellen Designers zu entwickeln [1]. Im Vergleich zu einer No-Code-Plattform soll bei einer LCP mit möglichst wenig Programmcode eine Software erstellt werden können.

Diese Arbeit beschäftigt sich ausschließlich mit der Erstellung der Anwendungslogik in einer LCP. Die Darstellungs- und Datenhaltungsschicht einer Software, die in einer LCP erstellt wird, wird nicht betrachtet. Die Grundlegende Idee dieser Arbeit ist, dass die Anwendungslogik einer Software nicht komplett in einer Bibliothek mittels Programmcode manuell programmiert wird, sondern in einzelne Bausteine zerlegt wird. Diese Bausteine können dann durch einen visuellen Designer zu einem Ablauf zusammengesetzt werden. Die Anwendungslogik ist somit nicht fest zur Kompilierzeit der Software einprogrammiert und kann zur Programmlaufzeit durch Abläufe mithilfe eines visuellen Designers modelliert werden.

Es wurde ein Konzept entworfen, welches ermöglicht Quellcode zu generieren, der die einzelnen Bausteine über Variablen und Parameter miteinander verbindet. Dieser Quellcode wird aus einem Modell eines Ablaufs generiert, kompiliert und ausgeführt. Die Bibliothek zur Erzeugung des Quellcodes, sowie der grafische Designer ist in C# mit .NET Core implementiert. Weitere Werkzeuge wie ein Debugger und Validierung der Abläufe ergänzen die Implementierung des Konzepts und erleichtern den Einsatz in der Praxis.

Als konkretes Beispiel für eine Verwendung der in dieser Arbeit erstellten Ablaufsteuerung wird ein Service Monitoring System verwendet. Das System hat die Aufgabe bestimmte Dienste auf einem Server zu überwachen, indem z.B. Logdateien und Statusinformationen der Services analysiert werden. Im Vergleich zu vielen anderen Monitoring Systemen, die nur überprüfen ob z.B. ein bestimmter Port geöffnet ist, kann dieses System den Zustand eines Dienstes genauer bestimmen.

2 Anforderungen

Ziel dieser Arbeit ist, ein Konzept für eine Ablaufsteuerung einer Low-Code-Plattform zu entwickeln und dieses dann zu implementieren. Dabei sollen folgende Anforderungen mitberücksichtigt werden:

- Ein Ablauf kann durch eine Administrationsoberfläche grafisch erstellt und bearbeitet werden.
- Einzelne Komponenten eines Ablaufs lassen sich wiederverwenden
- Änderungen eines Ablaufs sind möglich.

Diese Anforderungen sind entnommen aus [1].

Im Vergleich zu bereits existierenden LCP Anbietern, die ihre Plattform als Service in einer Cloud anbieten [2], ist diese Arbeit für lokale Rechner konzipiert. Diese Arbeit beschäftigt sich nicht mit der Modellierung von Arbeitsabläufen, sondern mit programminternen Abläufen bzw. Abläufen einer Software.

2.1 Anforderungen an die Bibliothek

- Plattformunabhängig: Unterstützung von gängigen Betriebssystemen Windows, Linux und macOS.
- Ein Ablauf kann serialisiert und deserialisiert werden, um einen einfachen Austausch über Dateien, sowie einfache Backups zu ermöglichen.
- Um die Handhabung des Systems zu vereinfachen und den Einstieg zu erleichtern, werden Variablen im Normalfall automatisch verwaltet.
- Ein inkonsistenter Ablauf kann verarbeitet aber nicht gestartet werden.
- Bei Änderung der Code-Creator können sich ggf. die Parameter verändern. Falls dieser Fall eintritt, gibt es die Möglichkeit einen inkonsistenten Ablauf zu reparieren.
- Es ist möglich, einen Ablauf als kompilierte .dll Datei zu exportieren.
- Abläufe können durch eine zentrale Stelle zwischen verschiedenen Rechnern ausgetauscht werden.
- Der Performance Overhead, der durch die Modellierung und Verarbeitung des Ablaufs entsteht, sollte möglichst gering sein und für einen Benutzer nicht wahrnehmbar sein.

2.2 Anforderungen an die Administrationsoberfläche

- Ein Ablauf kann visuell erstellt und bearbeitet werden.
- Alle Einstellungen, Argumente und Variablen lassen sich über die Oberfläche konfigurieren.
- Abläufe können über die Oberfläche manuell gestartet werden.
- Übersicht aller vorhandenen Abläufe, sowie Darstellung der aktuell und in der Vergangenheit ausgeführten Ablauf Instanzen.
- Inkonsistenzen zwischen Argumenten(-typen) und Variablen(-typen) werden dem Administrator als Fehler angezeigt.
- Nicht mehr vorhandene Aktivitäten werden durch einen Platzhalter ersetzt, damit der Administrator den Ablauf wiederherstellen kann.
- Elemente des Ablaufs können durch Anzeigenamen, Kommentare oder Farben hervorgehoben oder beschrieben werden.
- Es gibt einen Debugger, der Variablen und Parameter der Aktivitäten vor und nach dem Aufruf visuell darstellen kann.
- Fehlermeldungen der Bibliothek werden dem Benutzer angezeigt und helfen das Problem identifizieren zu können.
- Die Oberfläche soll möglichst einfach bedienbar und intuitiv sein, sodass ein Administrator nach einer kurzen Einarbeitung damit arbeiten und selbstständig einen Ablauf modellieren kann.

2.3 Anwendungsfälle

Die Ablaufsteuerung eignet sich im Allgemeinen sehr gut, um eine Middleware Software zu realisieren. Sie kann auch sinnvoll eingesetzt werden, wenn es bei einem Projekt möglich sein soll einzelne Bausteine der Software durch z.B. kundenspezifische Implementierungen ersetzten zu können.

Folgende Szenarien sind möglich:

2.3.1 Software in der Medizinbranche

Ein Unternehmen programmiert für mehrere verschiedene Krankenhäuser Softwarelösungen. Alle Krankenhäuser verwenden die im Medizinumfeld üblichen Standards, um z.B. Patienteninformationen auszutauschen. Obwohl Krankenhäuser die gleichen Standards verwenden, unterscheiden sie sich durch verschiedene Anforderungen an eine Software. Je nach Größe des Krankenhauses, muss die Softwareentwicklung verschiedene Aspekte beachten: Infrastruktur, Redundanz der Daten, Austausch der Daten mit dem Medizinischen Dienst der Krankenkassen, krankenhausspezifische Drittanbieter Software, Arbeitsabläufe, Anzahl der Patienten etc.

Durch die Verwendung einer Ablaufsteuerung kann die Software in einzelne Bausteine zerlegt werden. Bausteine, die standardisierte Daten verarbeiten, können für alle Krankenhäuser entwickelt und verwendet werden. Falls der Kunde spezielle Anforderungen an die Software hat, kann diese durch einen kundenspezifischen Baustein erweitert oder später einfach ergänzt werden. Mit dem Ansatz ein großes Problem in mehrere Bausteine zu zerlegen, kann ein komplexes System einfacher getestet und gewartet werden. Einzelne Komponenten lassen sich einfach abschalten oder austauschen.

2.3.2 Computerspielserver als Software as a Service

Es gibt viele Dienstleister, die einen Server für ein Computerspiel als SaaS anbieten. Dafür stellt der Dienstleister meistens eine Administrationsoberfläche zur Verfügung. Der Kunde kann über diese Oberfläche z.B. den Spielserver starten und stoppen. Ein Problem bei der Implementierung von so einer Administrationsoberfläche ist, dass die Spielserver unterschiedlich gehandhabt werden müssen: Bei Spielserver A kann der Prozess einfach beendet werden, bei Spielserver B muss „exit“ in die Konsole (Standard-Input) eingegeben werden, sodass dieser herunterfahren kann.

So ein Start- und Stopvorgang kann durch einen spielspezifischen Ablauf schnell modelliert werden. Ergänzend ist es möglich, diesen bei einem Update der Serversoftware zu aktualisieren.

2.3.3 Monitoring von Programmen

Es gibt viele Onlinedienste und Programme, die mittels TCP/UDP oder HTTP(S) Anfragen Programme überwachen. Eine Überwachung ob z.B. ein bestimmter Port geöffnet ist, stellt zwar die Uptime eines Systems dar, doch ist aus dieser Überwachung nicht ersichtlich, ob das System überhaupt korrekt funktioniert. Lediglich die Information, dass ein System nicht komplett abgestürzt ist, ist vorhanden.

Durch eine Ablaufsteuerung können programmspezifische Abläufe erstellt werden, die dann ein detaillierteres Ergebnis als einen Portzustand darstellen können. Zum Beispiel durchsucht ein Ablauf alle 10 Minuten die Logdateien eines Apache Webservers nach Fehlermeldungen. Falls Fehler gefunden werden, können diese dann an eine Übersicht weitergeleitet werden.

Solche Abläufe sind für jede Software individuell und müssen bei Änderung des zu überwachenden Programms gegebenenfalls angepasst werden. Aber einmal erstellt, kann ein Ablauf auf mehreren Systemen genutzt werden.

3 Verwandte Arbeiten

3.1 Microsoft Flow

In dem Buch „Introducing Microsoft Flow“ [3] beschreibt Vijai Anand Ramalingam Flow als ein Cloudservice von Microsoft, bei dem Abläufe über gängige Webbrowser erstellt und dann in der Cloud ausgeführt werden können. Der Fokus von diesem Service liegt dabei bei dem Datenaustausch zwischen verschiedenen Produkten. Über diverse Connectors können z.B. Informationen zwischen Office 365, Saleforce, RSS und Twitter ausgetauscht werden. Er beschreibt auch detailliert die Erstellung und Verwaltung der einzelnen Abläufe. In seiner Arbeit wird auch die Flow App vorgestellt, mit dieser Abläufe auf einem Mobiltelefon erstellt werden können.

Ein weiterer Connector ist der SharePoint Connector. Im Buch “Deploying SharePoint 2019: Installing, Configuring, and Optimizing for On-Premises and Hybrid Scenarios” [4] wird die Installation und Verwaltung eines SharePoint Servers beschrieben, aber auch die Integration von SharePoint und Microsoft Flow. Über das Microsoft On-Premises Data Gateway können Daten von einem in Microsoft Flow erstellten Ablauf und einem SharePoint Server ausgetauscht werden.

Eine Studie (beauftragt von Microsoft) von Forrester Consulting „The Total Economic Impact™ of PowerApps And Microsoft Flow” [5] stellt den großen Vorteil von Flow aus Perspektive der Finanzvorteile dar. Es wird erwähnt, dass sich eine Investition in Flow schon nach 3 Monaten amortisiert.

3.2 Microservices

Es gab und gibt unter Entwicklern viele Diskussionen um das Thema Microservices. In einem Artikel schreibt Alexander Schwartz, dass Internetriesen wie Amazon und Netflix zeigen, wie man ein Produkt veröffentlicht und dass die Microservice Architektur auch in der Praxis funktioniert. Jedes 5-10 Personen große Team entwickelt und betreibt einen Microservice. – Zusammen ergibt das dann eine komplexe Software [6].

Auf der Webseite JAXenter wird mit dem Beitrag „Wie lässt sich Ordnung in einen Haufen (Micro-)Services bringen?“ die Microservice Architektur in Frage gestellt. Natürlich haben Microservices Vorteile, die die großen Firmen so erfolgreich gemacht haben. Aber Routing, Message Queues und Eventbus können zusätzliche Probleme mit sich bringen: Was ist, wenn ein Service ausfällt? Auch wird beschrieben, dass z.B. bei einem Bestellprozess der Zustand einer Bestellung nur durch die Abfrage mehrerer Microservices bestimmen lässt. Ein weiterer Vorschlag in diesem Artikel ist, einen Microservice mit einer Workflow-Engine und BPM umzusetzen. Da ein Microservice immer eine Aufgabe übernimmt, kann diese auch in einem Ablauf modelliert werden. Doch auch mit diesem Ansatz ist das Problem der Kommunikation zwischen den Microservices nicht behoben. Im Artikel wird alternativ vorgeschlagen, ein „zentraler Workflow-Microservice, der die Steuerung aller Geschäftsprozesse übernimmt“ zu verwenden, um die Kommunikation zu erleichtern [7].

Im codecentric Blog antwortet Tobias Flohre, dass der Ansatz mit einem zentralen Workflow-Microservice – je nach Definition – ggf. gar keine Microservice Architektur mehr ist. Gleichzeitig stellt er in seinem Beitrag „Wer Microservices richtig macht, braucht keine Workflow Engine und kein BPMN“ einen Microservice Ansatz dar, um den Bestellprozess mit Microservices ohne zentrale Stelle zu realisieren [8].

Zusammenfassend kann man in den Kommentaren des Blogeintrags lesen, dass beide Lösungen ihre Daseinsberechtigung haben und es auf die Anforderungen und die Skalierung ankommt, welche Lösung nun besser geeignet ist, um ein bestimmtes Problem zu lösen.

4 Grundlagen

4.1 Low-Code-Platform

Das Unternehmen Forrester Research [9] definiert eine LCP folgendermaßen: „Platforms that enable rapid delivery of business applications with a minimum of hand-coding and minimal upfront investment in setup, training, and deployment.” [9]. In diesem zitierten Artikel ist zusätzlich definiert, dass in einer Entwicklungsumgebung das Datenmodell visuell definiert wird. Somit wird kein klassisch entwickelter Programmcode benötigt, um Datenbanken oder Webservices anzusprechen. Die Anwendungslogik wird visuell über Abläufe, Entscheidungstabellen und Geschäftsregeln erstellt, um auch in dieser Schicht keinen bzw. wenig Programmcode entwickeln zu müssen. Des Weiteren wird bei einer LCP laut Forrester auch die Präsentationsschicht über eine grafische Oberfläche mittels Drag and Drop und einzelnen Komponenten erstellt. Zusätzlich muss eine LCP Unterstützung zum Testen der Software und zum Deployvorgang bereitstellen [9]. Zusammengefasst ist eine LCP eine umfangreiche Entwicklungsumgebung, um damit eine Software vom Datenmodell bis zur Präsentationsschicht visuell erstellen und veröffentlichen zu können.

4.2 ASP.NET Core

Das Framework ASP.NET Core ist plattformunabhängig, hoch performant und Open Source. Es kann benutzt werden um z.B. Web-Apps und Services, IoT Apps und Backends für mobile Anwendungen zu erstellen [10]. Das alte ASP.NET, welches 1998 veröffentlicht wurde, ist mit ASP.NET Core 2016 überholt worden. Dabei wurden die besten Bestandteile von ASP.NET und dessen Erweiterungen in ein gemeinsames Framework zusammengefügt. ASP.NET Core ist kompatibel zum .NET Framework und .NET Core [11].

4.3 .NET Compiler Platform

Der auch unter dem Codenamen Roslyn bekannter Compiler wurde 2011 von Microsoft veröffentlicht. Dieser neu entwickelte Compiler kann als Dienst betrachtet werden, mit dessen es möglich ist, bei jedem Schritt der Kompilierung mit dem Zwischenergebnis zu arbeiten [12]. Die Roslyn Pipeline beschreibt die hintereinander Ausführung einzelner Schnittstellen, um Quellcode zu einem Assembly zu kompilieren. Über eine Schnittstelle kann z.B. der Quellcode zuerst geparst und dann semantisch analysiert werden [13].

Dadurch ist Roslyn im Vergleich zum vorherigen Compiler keine Black-Box mehr [13]. Mit der Symbol API wird beispielsweiße der Datentyp eines Parameters kontextbezogen bestimmt und über die Emit API wird ein Assembly erzeugt, welches dann geladen und verarbeitet werden kann.

4.4 LiteDB

LiteDB ist eine Open Source NoSQL Datenbank welche komplett in C# entwickelt wurde. Sie ist .NET Standard kompatibel und somit auch kompatibel zu .NET Core. Diese Datenbank ist serverlos und speichert die Daten in einer Datei ab. Die LiteDB wird bei einem Projekt als Bibliothek referenziert und mittels einer MongoDB ähnlichen Schnittstelle können die Tabellen bearbeitet werden. [14]. Sie ist vergleichbar mit der bekannteren SQLite Datenbank.

5 Konzept

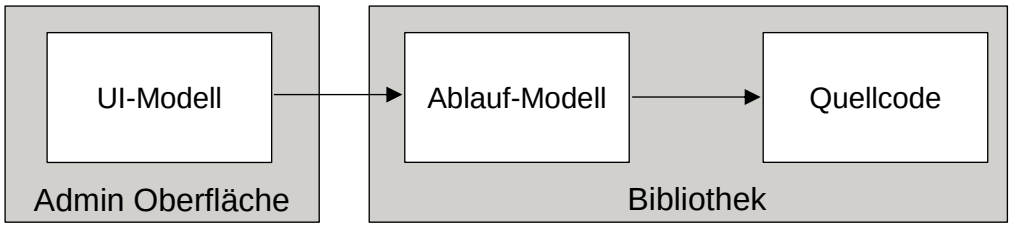

Abbildung 1: Konzept vereinfachte Übersicht

Der Anwender benötigt eine Oberfläche, die einen grafischen Editor zur Verfügung stellt. Mit diesem Editor können Abläufe erstellt und modifiziert werden. Zusätzlich stellt die Oberfläche Informationen über derzeit laufende oder in der Vergangenheit ausgeführte Instanzen der Abläufe dar.

Des Weiteren wird eine Bibliothek benötigt, die die eigentliche Ausführung des Ablaufs übernimmt.

Das Konzept sieht vor, dass die Administrationsoberfläche und die eigentliche Bibliothek separierbar sind, sodass die Bibliothek Abläufe ohne Editor ausführen und verarbeiten kann. Damit ist es möglich, diese in ein vorhandenes Projekt einfacher zu integrieren.

Wenn der Benutzer auf der Administrationsoberfläche einen Ablauf bearbeitet, wird das UI-Modell in das Ablauf-Modell konvertiert und der Bibliothek übergeben. Im nächsten Schritt wird von der Bibliothek aus dem Ablauf-Modell Quellcode erzeugt. Dieser erzeugte Code wird zur Laufzeit kompiliert und kann dann effizient ausgeführt werden.

Es werden unterschiedliche Modelle eines Ablaufs benötigt, da es möglich sein soll, einen Ablauf mit zusätzlichen Metadaten, wie z.B. Kommentaren und weiteren grafischen Elementen, zu versehen. Diese Attribute des UI-Modells werden ausschließlich für den grafischen Editor benötigt und beinhalten keine Informationen zum eigentlichen Ablauf. Sie werden als Metadaten im Ablauf-Modell gespeichert, jedoch müssen sie von der Bibliothek nicht verarbeitet werden. Dies vereinfacht die weitere Verarbeitung des Ablaufs.

5.1 Administrationsoberfläche

Diese Oberfläche stellt alle Funktionen zur Verfügung, die benötigt werden, um einen Ablauf zu erstellen und zu modifizieren. Ein Ablauf lässt sich durch diese Oberfläche starten und stoppen. Zusätzlich sollen Informationen von den aktuell ausgeführten Instanzen oder in der Vergangenheit ausgeführten Instanzen dargestellt werden.

Die Administrationsoberfläche wird mittels einer Webanwendung realisiert.

Vorteile:

- Es wird keine Clientsoftware benötigt
- Das Betriebssystem des Clients ist nicht relevant
- Ein System kann durch die Webseite von „außen“ modifiziert werden

Nachteile:

- Eine konsistente Darstellung zwischen verschiedenen Browsern kann kompliziert zu implementieren sein.
- Eine Webanwendung muss zusätzliche Sicherheitsanforderungen wie Authentisierung und Autorisierung erfüllen, da Webanwendungen meistens über das Internet aufrufbar sind.

Ein Webserver verwaltet das UI-Modell und generiert HTML aus diesem Modell, sodass ein Browser den Ablauf in einer, für den Benutzer geeigneten Weiße, darstellen kann. Falls der Anwender diesen Ablauf im Browser editiert, muss auch das UI-Modell serverseitig durch den Webserver angepasst werden. Um den Ablauf abspeichern oder starten zu können, wird dieser in das Ablauf-Modell konvertiert, indem die Informationen zur Darstellung des Ablaufes in Metadaten das Ablauf-Modells konvertiert werden.

5.1.1 UI-Modell

Dieses Modell repräsentiert den aktuellen Zustand eines Ablaufes in der Web-Ansicht. Im Vergleich zum Ablauf-Modell (siehe 5.2.1) enthält auch das UI-Modell die Struktur und die Variablen des Ablaufs, besitzt aber zusätzliche Attribute, die für die Darstellung des Ablaufs mittels HTML benötigt werden.

5.2 Bibliothek

Die Hauptaufgabe der Bibliothek ist die Konvertierung des Ablauf-Modells zu Quellcode. Der generierte Quellcode wird kompiliert und ausgeführt. Des Weiteren übernimmt die Bibliothek die persistente Speicherung der Abläufe, sowie die Serialisierung. Durch die Serialisierung eines Ablaufs, kann dieser z.B. in einer Textdatei abgespeichert oder über Netzwerk ausgetauscht werden.

5.2.1 Ablauf-Modell

Das Ablauf-Modell beinhaltet alle benötigten Informationen, um aus diesem Modell Quellcode erzeugen zu können. Die einzelnen Bausteine eines Ablaufs, die zusammengesetzt den eigentlichen Ablauf definieren, werden durch Code-Creator repräsentiert. Ergänzend enthält das Ablauf-Modell die Parameter des Ablaufs, referenzierte Bibliotheken, referenzierte Namespaces, sowie die Metadaten aus dem UI-Modell.

5.2.2 Code-Creator

Ein Ablauf kann aus einem oder mehreren Code-Creator bestehen. Ein Code-Creator ist eine Klasse, die Quellcode erzeugt. Die Erzeugung des Quellcodes ist meistens abhängig von bestimmten Parametern, die über den Ablauf Editor oder durch den Kontext, an welcher Stelle sich der Code-Creator im Ablauf befindet, festgelegt werden.

Ein Code-Creator kann auch optional ein Code-Creator-Container sein. Dieser Container ist ein Code-Creator mit der zusätzlichen Eigenschaft, dass er weitere Code-Creator als Kinder beinhalten kann. Dadurch können z.B. Sequenzen und Schleifen realisiert werden.

5.2.3 Activity

Eine Activity ist eine Klasse, die im Gegensatz zu einem Code-Creator keinen Quellcode für den eigentlichen Ablauf erzeugt. Sie beinhaltet aber eine Implementierung, die von einem Ablauf aufgerufen werden kann. Dazu muss es einen Code-Creator geben, der den Quellcode für den Aufruf einer Activity in einem Ablauf generiert.

Eine Activity ist somit eine vereinfachte Form eines Bausteins eines Ablaufs. Sie wird in der Praxis oft eingesetzt, da es in vielen Anwendungsfällen nicht nötig ist, den Quellcode des Ablaufs direkt zu modifizieren.

Eine Implementierung könnte folgendermaßen aussehen:

```
public class ConsoleWriteLineActivity : ICodeActivity
{
    public void Execute(string pText)
    {
        Console.WriteLine(pText);
    }
}
```

6 Implementierung des Konzeptes

Um das in den Anforderungen definierte Plattformunabhängigkeits-Kriterium möglichst gut erfüllen zu können, muss das Konzept in einer plattformunabhängigen Programmiersprache implementiert werden. Das Konzept setzt zusätzlich voraus, dass zur Laufzeit Code kompiliert und dieser dann ausgeführt werden kann. Diese Arbeit ist in C# mit .NET Core implementiert. Alternativ könnte z.B. Java verwendet werden.

Die mit dieser Arbeit verbundene Implementierung des Konzepts trägt den Namen Coreflow. „Core“ bezieht sich auf .NET Core und dem Kern einer Software. „Flow“ beschreibt einen Ablauf. Zusammengesetzt beschreibt der Name zusätzlich einen „Fließenden Kern“ – Also einer der funktioniert und im „Flow“ ist. Coreflow.Web ist die Implementierung der Administrationsoberfläche.

6.1 Administrationsoberfläche: Coreflow.Web

Die Administrationsoberfläche wird laut Konzept in einem Browser dargestellt. Sie kann in die zwei Bestandteile Webserver und Webseite aufgeteilt werden.

6.1.1 Webserver

Der Webserver ist mittels ASP.NET Core und dem Model-View-Controller Entwurfsmuster umgesetzt. Die Benutzer Authentisierung ist bereits im „ASP.NET Core Identity“ Packet von Microsoft standardmäßig enthalten. Diese wird für den Login des Administrators genutzt. Die Datenhaltung für die Benutzerinformationen ist mit einer LiteDB implementiert. Für die Abläufe und Instanzen nutzt Coreflow.Web die Datenhaltung der Coreflow Bibliothek. Da der Webserver mit dem UI-Modell arbeitet und Coreflow mit dem Ablauf-Modell, konvertiert der Webserver das Modell je nach Bedarf in beide Richtungen. Auch Ablauf Instanzen und Validierungsfehler bereitet der Webserver auf, damit diese auf der Webseite dargestellt werden können. Er verfügt auch über einen temporären Ablauf Speicher. Dadurch sind ungespeicherte Veränderungen eines Ablaufes temporär gespeichert, auch wenn das Browserfenster geschlossen wird. Zusätzlich kann die temporäre Version eines Ablaufs verworfen werden, falls ungewollte Veränderungen an einem Ablauf durchgenommen wurden.

6.1.2 Webseite

Die Administrationsoberfläche besteht hauptsächlich aus dem Ablauf Editor und den Informationsseiten über die Instanzen der Abläufe. Sie werden mittels HTML und CSS im Browser dargestellt. Interaktionen des Editors, die mit dem Webserver synchronisiert werden müssen, sind mit Javascript (JQuery) und Ajax umgesetzt und bearbeiten das in Coreflow.Web temporär gespeicherte UI-Modell eines Ablaufs.

Ablauf Editor

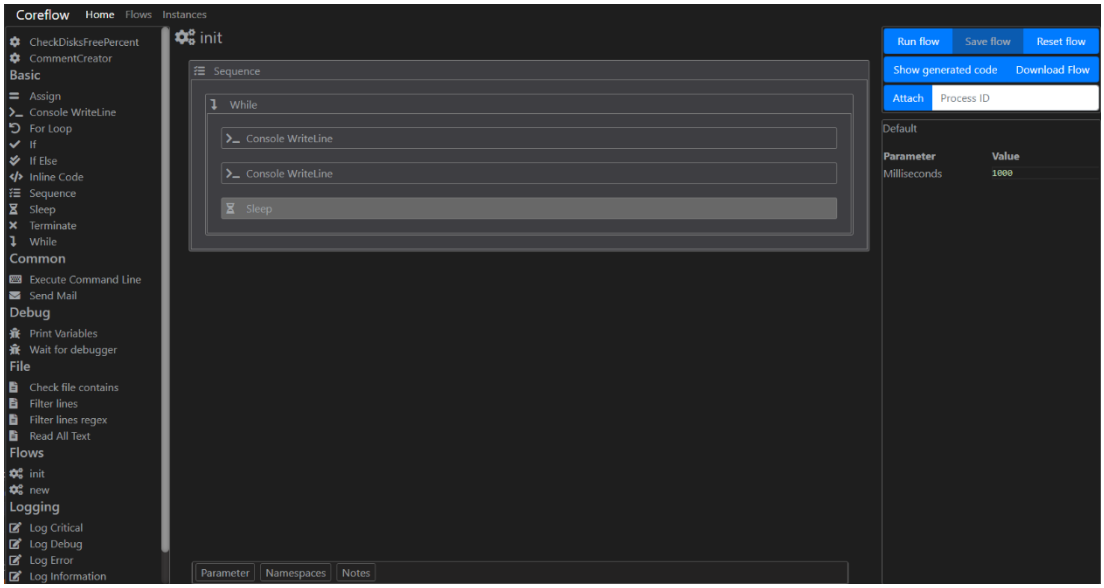

Abbildung 2: Ablaufeditor

Auf der linken Seite des Ablaufeditors sind alle Code-Creator zu sehen, die standardmäßig oder durch ein Plugin vorhanden sind. Mittels Drag and Drop kann ein Code-Creator dem Ablauf in der Mitte hinzugefügt werden. Selbstverständlich kann auch ein bereits erstellter Code-Creator verschoben oder gelöscht werden. Durch einen Mausklick auf einen Code-Creator wird dieser ausgewählt und es werden im rechten Fenster die Parameter angezeigt, die angepasst werden können. Über einen Doppelklick auf ein Textfeld der Parameter kann der Inhalt in einer größeren Ansicht des Editors angezeigt werden. Diese Arbeit nutzt den von Microsoft implementierten und MIT lizensierten Monaco Editor, der für Visual Studio Code entwickelt wurde [15]. Durch das integrierte Syntax-Highlighting können auch komplexere C# Codefragmente anschaulich dargestellt werden. Im unteren Bereich des Ablaufeditors können

Parameter des Ablaufs mit Standardwerten, sowie die zu benutzenden C# Namespaces definiert werden.

Zusätzlich ist es möglich Code-Creator mit Notizen und Farben zu versehen. Somit können Code-Creator markiert werden, um dem Administrator die Modellierung eines Ablaufs zu erleichtern (siehe 7.7).

6.2 Bibliothek: Coreflow

Umgesetzt ist die Bibliothek durch eine C# .NET Core Klassenbibliothek. Der Hauptbestandteil sind einige Interfaces zur Modellierung des Ablauf-Modells, die Implementierung der Konvertierung vom Ablaufmodell zum Quellcode, sowie die Kompilierung des Quellcodes. Durch die Implementierung mit C# .NET Core kann diese Bibliothek auf Windows, macOS und vielen Linuxdistributionen wie Debian, Ubuntu, CentOS und openSUSE verwendet werden.

6.2.1 Serialisierung des Ablauf-Modells

Um das Ablauf-Modell speichern oder exportieren zu können, wird dieses in ein XML serialisiert. Da der mitgelieferte XML Serializer in .NET Core (von Microsoft) komplexe Objektstrukturen mit mehreren Interfaces und Vererbung derzeit nicht korrekt verarbeitet, wird der „Extended XML Serializer“ [16] verwendet. Für diese Bibliothek wurden für bestimmte Objekte „Custom Serializer“ implementiert, sodass auch Klassen, die nicht mit der Standardimplementierung serialisiert werden können, korrekt serialisiert werden. Dadurch kann dieser Serializer das Ablauf-Modell serialisieren und wieder deserialisieren.

6.2.2 Konvertierung des Ablauf-Modells zu Quellcode

Ein wichtiger Schritt der Verarbeitungskette ist die Konvertierung des Ablaufs zum eigentlichen Quellcode.

Aus einem Ablauf-Modell wird eine Klasse erzeugt. Um den Ablauf später einfach aufrufen zu können, implementiert diese Klasse das ICompiledFlow Interface, welches eine „Run“ Methode definiert. Argumente des Ablaufs, sowie der Code-Creator werden durch Membervariablen dargestellt. Die Membervariable „InstanceId“ ist bei einem Ablauf immer vorhanden. Dadurch erhält eine Instanz des Ablaufs immer eine eindeutige Bezeichnung.

Die Implementierung der „Run“ Methode wird durch die Code-Creator des Ablauf-Modells generiert. Dabei ruft der Ablaufgenerator die „ToCode“ Methode von jedem Code-Creator im Ablaufmodell in der richtigen Reihenfolge und Verschachtelung auf. Dadurch erzeugt jeder Code-Creator ein Codefragment an der vorgesehenen Stelle und es entsteht die Implementierung des Ablaufs. Variablen des Ablaufmodells werden durch lokale Variablen in der „Run“ Methode repräsentiert.

Aufgrund der Anforderung, dass Variablentypen automatisch erkannt und verarbeitet werden, muss die Konvertierung berücksichtigen, ob eine bestimmte Variable im aktuellen Geltungsbereich vorhanden ist oder nicht. Doch damit der Quellcode semantisch verarbeitet werden kann und die aktuell definierten Variablen bestimmt werden können, muss dieser syntaktisch korrekt sein. Deswegen wird der Quellcode von außen nach innen generiert. Somit ist sichergestellt, dass zu (fast) jedem Zeitpunkt der Quellcode von Roslyn semantisch analysiert werden kann.

Diese Konvertierung ist in vielen Fällen sehr komplex und sie ist abhängig von den verwendeten Code-Creator in einem Ablauf. Es ist dadurch für mich nicht möglich eine abstrakte allgemeingültige Beschreibung der Konvertierung anzugeben. Auch wird in dieser Arbeit nicht bewiesen, dass diese Konvertierung immer korrekt funktioniert. Doch durch diverse Tests konnte diese Umwandlung so optimiert werden, dass es derzeit keine bekannten Konstellationen gibt, die nicht konvertiert werden können.

Beispiel einer Konvertierung vom Ablauf-Modell zum Quellcode

Der in diesem Beispiel genutzte Ablauf besteht aus einer Sequenz, die eine If Else Abfrage beinhaltet. Die If Abfrage überprüft, ob die Stunde der aktuellen Uhrzeit 0 entspricht. Falls ja wird „Es ist Mitternacht!“ ausgegeben. Falls es nicht Mitternacht ist, wird die aktuelle Stunde ausgegeben.

Grafische Darstellung des zu konvertierenden Ablaufs (zum Verständnis):

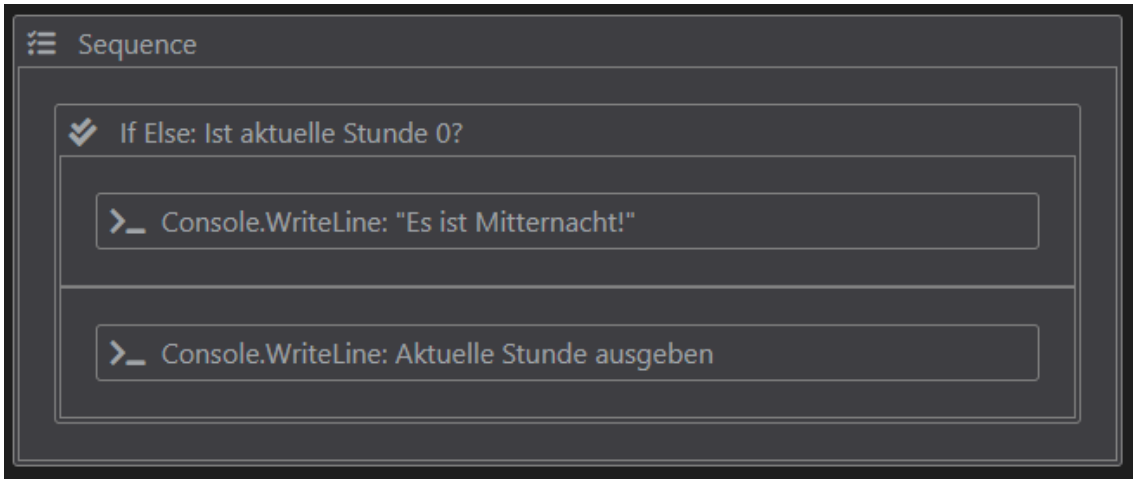

Abbildung 3: Grafische Darstellung des Beispielablaufs

Vereinfachte XML Darstellung des Ablauf-Modells

Das Ablauf-Modell beinhaltet alle Informationen, um daraus den Quellcode erzeugen zu können. Das Modell beinhaltet die Struktur des Ablaufs, sowie die Informationen zu den Parametern der jeweiligen Code-Creator.

```
<FlowDefinition>
  <ReferencedNamespaces>[…]</ReferencedNamespaces>
  <ReferencedAssemblies>[…]</ReferencedAssemblies>
  <CodeCreator exs:type="ns1:SequenceCreator">
    <CodeCreators>
      <sys:List exs:arguments="ICodeCreator">
        <ns1:IfElseCreator>
          <CodeCreators>
            <sys:List exs:arguments="ICodeCreator">
              <ns1:CodeActivityCreator
                    exs:arguments="ConsoleWriteLineActivity">
                <Arguments>
                  <ns2:InputExpressionCreator>
                    <Code>"Es ist Mitternacht!"</Code>
                    <Name>pText</Name>
                    <Type>System.String</Type>
                  </ns2:InputExpressionCreator>
                </Arguments>
              </ns1:CodeActivityCreator>
            </sys:List>
            <sys:List exs:arguments="ICodeCreator">
              <ns1:CodeActivityCreator
                    exs:arguments="ConsoleWriteLineActivity">
                <Arguments>
                  <ns2:InputExpressionCreator>
                    <Code>"Aktuelle Stunde: " +
                                  DateTime.Now.Hour</Code>
                    <Name>pText</Name>
                    <Type>System.String</Type>
                  </ns2:InputExpressionCreator>
                </Arguments>
              </ns1:CodeActivityCreator>
            </sys:List>
          </CodeCreators>
          <Arguments>
            <ns2:InputExpressionCreator>
              <Code>DateTime.Now.Hour == 0</Code>
              <Name>Expression</Name>
              <Type>Coreflow.Interfaces.CSharpCode</Type>
            </ns2:InputExpressionCreator>
          </Arguments>
        </ns1:IfElseCreator>
      </sys:List>
    </CodeCreators>
  </CodeCreator>
</FlowDefinition>
```

Schritt 1

Da der Quellcode von außen nach innen generiert wird, existiert bei der Generierung immer ein oberer und unterer Teil des Quellcodes. Ein Code-Creator fügt meistes dem oberen oder unteren Quellcodeteil Programmcode hinzu.

In diesem Beispiel ist der aktuell obere Quellcodeteil grau hinterlegt.

Zuerst wird der Namespace, die Klasse, sowie die „Run“ Methode generiert.

```
namespace FlowNs___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219
{
  public class Flow___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219 : ICompiledFlow
  {
    public Guid InstanceId = Guid.NewGuid();

    public void Run()
    {
    }
  }
}
```

Schritt 2

Nun werden alle Code-Creator bestimmt, die im Aktuellen und im nächst tieferem verschachteltem Bereich (z.B. Sequenz) vorhanden sind. In diesem Beispiel ist dies die „ConsoleWriteLineActivity“. Der passende Code-Creator erzeugt nun den benötigten Initialisierungscode für die Activity. Dadurch, dass die Erzeugung des „ConsoleWriteLineActivity“ Objekts eine Verschachtelungsebene vor der ersten Verwendung erzeugt wird, kann das Objekt gegebenenfalls öfters verwendet werden und muss deshalb nicht immer neu erzeugt werden (siehe 8.2). In diesem Fall wird dasselbe Objekt im positiv oder negativ Fall der if Abfrage verwendet. Siehe dazu Schritt 3 ff.

```
namespace FlowNs___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219
{
  public class Flow___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219 : ICompiledFlow
  {
    public Guid InstanceId = Guid.NewGuid();

    public void Run()
    {
      ConsoleWriteLineActivity _cc_9bf4d5f6_c6f2_4b1d_a10a_cf55fd48f9ec
                                              = new ConsoleWriteLineActivity();
    }
  }
}
```

Schritt 3

Das Argument der If Abfrage wird anhand des Ablauf-Modells eingefügt. Daraufhin wird der Aufruf der „Execute“ Methode mit den passenden Argumenten ergänzt.

```
namespace FlowNs___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219
{
  public class Flow___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219 : ICompiledFlow
  {
    public Guid InstanceId = Guid.NewGuid();

    public void Run()
    {
      ConsoleWriteLineActivity _cc_9bf4d5f6_c6f2_4b1d_a10a_cf55fd48f9ec
                                              = new ConsoleWriteLineActivity();
      if (
      DateTime.Now.Hour == 0
      )
      {
        __cc_9bf4d5f6_c6f2_4b1d_a10a_cf55fd48f9ec.Execute(
        "Es ist Mitternacht!"
        );
      } // "Problemklammer"
    }
  }
}
```

Nun liegt ein Problem in der Codegenerierung vor. Die „Problemklammer“ gehört zum unteren Teil des Quellcodes. Dies ist für den ersten bereits erzeugten Bereich der If Else Abfrage nötig, da sonst der tiefer geschachtelte Quellcode des ersten Bereichs nicht erzeugt werden kann. Damit aber der zweite („else“) Bereich der If Else Abfrage erzeugt werden kann, muss diese Klammer in den oberen Quellcode Teil verschoben werden. Man könnte diese Verschiebung auch als die Verschiebung der Position des Cursors betrachten.

Schritt 4

Das in Schritt 3 beschriebene Problem wird behoben, indem der entsprechend erzeugte Code (In diesem Fall die „Problemklammer“) aus dem unteren Bereich entfernt und dem oberen Bereich hinzugefügt wird. Dieses Umhängen ist immer bei Code-Creator nötig die mehrere Codebereiche erzeugen können. Um die Implementierung eines Code-Creators mit zwei Bereichen zu vereinfachen kann die Klasse AbstractDualSequenceCreator verwendet werden werden.

Nach dem Umhängen befindet sich die „Problemklammer“ im oberen Bereich des generieten Quellcodes.

```
namespace FlowNs___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219
{
  public class Flow___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219 : ICompiledFlow
  {
    public Guid InstanceId = Guid.NewGuid();

    public void Run()
    {
      ConsoleWriteLineActivity _cc_9bf4d5f6_c6f2_4b1d_a10a_cf55fd48f9ec
                                              = new ConsoleWriteLineActivity();
      if (
      DateTime.Now.Hour == 0
      )
      {
        __cc_9bf4d5f6_c6f2_4b1d_a10a_cf55fd48f9ec.Execute(
        "Es ist Mitternacht!"
        );
      } // "Problemklammer"
    }
  }
}
```

Schritt 5

Das „else“ Schlüsselwort, sowie der zweite innere Bereich der If Else Abfrage wird nun hinzugefügt. Dabei wird auch, wie im ersten Bereich, das bereits eine Ebene darüber erzeugte „ConsoleWriteLineActivity“ Objekt verwendet.

Die Erzeugung des Quellcodes ist nun abgeschlossen.

```
namespace FlowNs___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219
{
  public class Flow___cc_aa0aba2a_6b96_44ff_ba72_1d77869b7219 : ICompiledFlow
  {
    public Guid InstanceId = Guid.NewGuid();

    public void Run()
    {
      ConsoleWriteLineActivity _cc_9bf4d5f6_c6f2_4b1d_a10a_cf55fd48f9ec
                                              = new ConsoleWriteLineActivity();
      if (
      DateTime.Now.Hour == 0
      )
      {
        __cc_9bf4d5f6_c6f2_4b1d_a10a_cf55fd48f9ec.Execute(
        "Es ist Mitternacht!"
        );
      }
      else
      {
        __cc_9bf4d5f6_c6f2_4b1d_a10a_cf55fd48f9ec.Execute(
        "Aktuelle Stunde: " + DateTime.Now.Hour
        );
      }
    }
  }
}
```

Hinweis: Der in diesen Schritten beschriebene Quellcode wurde vereinfacht. Namespace Angaben, „using“ Anweisungen, geschweifte Klammern für die äußere Sequenz, sowie Kommentare für die Fehlerbehandlung wurden zur übersichtlicheren Darstellung weggelassen.

6.2.3 Aufruf eines anderen Ablaufs

Damit ein Ablauf einen anderen Ablauf aufrufen kann, wird für jeden vorhandenen Ablauf ein Code-Creator erzeugt, mithilfe dessen der Quellcode für einen Aufruf eines Ablaufs erzeugt wird. Bei einem Aufruf werden die Membervariablen, sowie vom Benutzer definierte Argumente des aktuellen Ablaufs dem aufzurufenden Ablauf übergeben.

Dieses Verhalten vermittelt dem Benutzer, dass ein Ablauf auch wieder ein Code-Creator ist. Dies ist zwar technisch nicht korrekt, da ein Ablauf ein tatsächliches Interface implementiert und selbst keinen Quellcode erzeugt, doch es wird angenommen, dass dieses technische Detail nur für einen Programmierer relevant ist und somit aus Anwendersicht diese Annahme korrekt ist.

6.2.4 Fehlerbehandlung bei Kompilierfehler

Aufgrund von ungültigen Eingaben durch den Anwender kann es vorkommen, dass aus einem Ablauf kein gültiger Quellcode generiert werden kann. Falls dieser Fall eintritt, zeigt die Administrationsoberfläche das Problem an der entsprechenden Stelle an. Um eine Quellcodezeile einem Parameter zuzuordnen, werden während der Codegenerierung Kommentare mit den IDs der generierten Parameter hinzugefügt. Wenn der Compiler nun einen Fehler in z.B. Zeile 12 findet, wird die zugehörige ID des Parameters im Quellcode gesucht. Die Administrationsoberfläche kann dadurch den Fehler einem Textfeld zuordnen und somit den Fehler für den Anwender eingrenzen.

6.2.5 Kompilierung des generierten Quellcodes

Der generierte Quellcode wird, durch den in .NET Core integrierten Compiler Roslyn kompiliert. Alle Abläufe werden in einem gemeinsamen Assembly kompiliert, damit sich die Abläufe gegenseitig aufrufen können, ohne zusätzliche Referenzen angeben zu müssen. Da sich jeder Ablauf in einem eigenen Namespace befindet, können Konflikte ausgeschlossen werden. Das erzeugte Assembly wird daraufhin in den aktuellen Prozess geladen. Jetzt kann der generierte Code und somit der Ablauf ausgeführt werden.

6.2.6 Export der kompilierten Abläufe

Ein zuvor generiertes Assembly kann exportiert werden, sodass ein kompilierter Ablauf auch ohne die Coreflow Bibliothek verwendet werden kann. Dabei ist zu beachten, dass .NET Core zwischen Implementation und Contract assemblies unterscheidet. Die Implementation Assemblies sind die, die zur Laufzeit tatsächlich verwendet werden. Die Contract Assemblies definieren die Schnittstelle zu den Implementation Assemblies [17]. Wenn nun ein Ablauf kompiliert werden soll, sollte dieser mit den Contract Assemblies gelinked werden. Beim Kompilieren des Quellcodes werden deshalb die Implementation Assemblies mit den Contract Assemblies ersetzt.

Dieser Vorgang ist erforderlich, damit ein exportierter Ablauf als Bibliothek genutzt werden kann. Eine Entwicklungsumgebung wie Visual Studio bzw. deren Compiler kompiliert Quellcode im Normalfall mit den Contract Assemblies.

7 Weitere implementierte Werkzeuge

Während der Implementierung des Konzepts stellte sich heraus, dass zusätzliche Programmteile benötigt werden, um die Ablaufsteuerung praxistauglich zu machen. Diese Werkzeuge haben nichts mit dem Konzept und der eigentlichen Codegenerierung zu tun, erweitern aber die Ablaufsteuerung, um diese in der Praxis einsetzen zu können.

7.1 Coreflow Host

Mit dieser Konsolenanwendung können Abläufe ohne den Webserver (Coreflow.Web) gestartet werden. Sie referenziert die Bibliothek (Coreflow), kompiliert beim Start automatisch alle Abläufe und startet einen Ablauf mit dem Namen „init“. Somit ist es möglich einen Ablauf auf einem System mit Coreflow.Web zu entwickeln und diesen dann auf einem anderen System auszuführen. Gleichzeitig ist diese Anwendung ein Beispiel für eine trivial Implementierung der Coreflow Bibliothek.

7.2 Flow Repository

Um Abläufe zwischen mehreren Systemen zu synchronisieren kann das Flow Repository verwendet werden. Durch einen Webservice können die Serialisierten Abläufe von beliebigen Clients abgerufen oder bearbeitet werden. Somit kann ein Ablauf mit einer Coreflow.Web Installation erstellt und von mehreren Coreflow.Host Installationen ausgeführt werden.

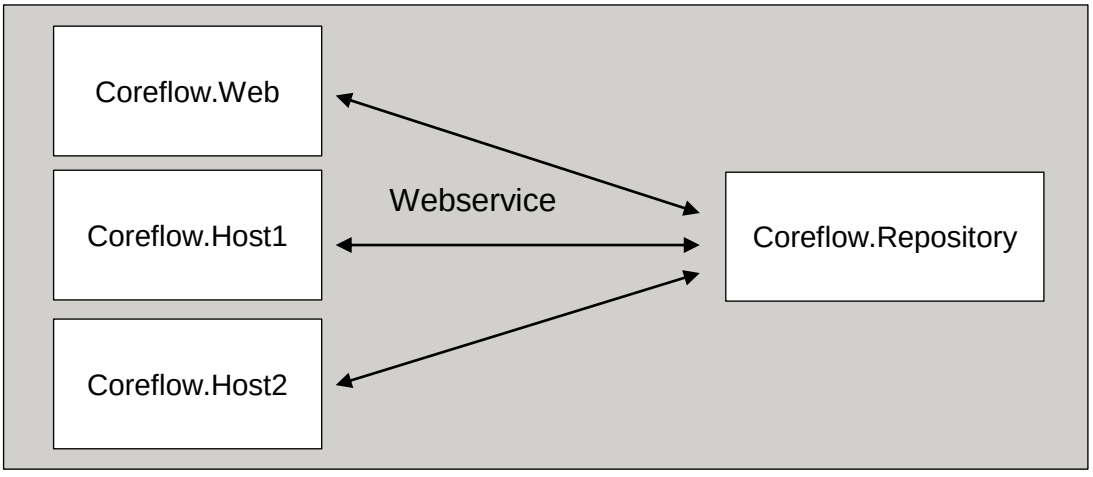

Abbildung 4: Anbindung Flow Repository

7.3 Debugger für Abläufe

Während der Entwicklung dieses Features wurde bekannt, dass es nicht möglich ist, einen Breakpoint außerhalb von Visual Studio bzw. ohne einen Debugger an eine Software anzuhängen. Auch ist es sehr kompliziert, wenn ein Prozess sich selbst debuggt, da dies in vielen Fällen zu Deadlocks führen kann. Ein Deadlock tritt z.B. ein, wenn der Debugger alle Threads eines Prozesses blockiert. Da der Fokus dieser Arbeit nicht auf der Implementierung eines Debuggers liegt muss ein externer Debugger verwendet werden, um dieses Feature zu ermöglichen. Es liegt nahe, den integrierten Debugger von Visual Studio zu verwenden, doch Microsoft hat diesen nicht als Open Source Komponente von .NET Core veröffentlicht und verbietet zusätzlich ausdrücklich die Verwendung in nicht Microsoft Produkten [18]. Weitere Informationen über das Verbot in Anhang A.

Aufgrund dieser einschränkenden Lizenzbestimmungen wird in dieser Arbeit der Debugger „netcoredbg“ benutzt [19].

Der Debugger ist folgendermaßen in diese Arbeit integriert:

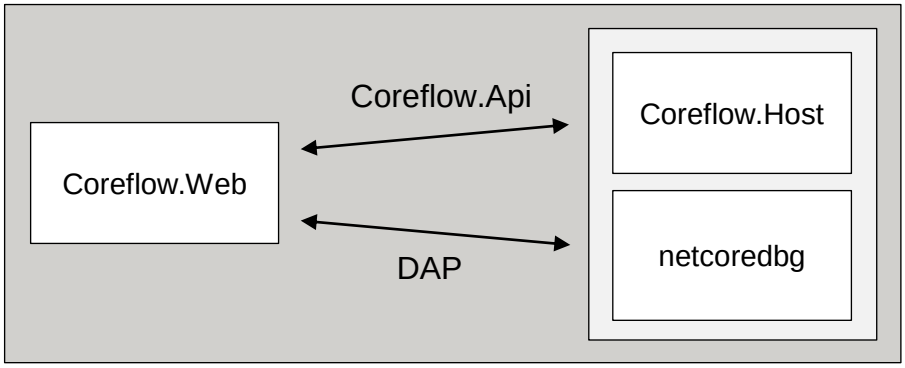

Abbildung 5: Integration des Debuggers

Coreflow.Web benutzt das Debug Adapter Protocol (DAP), welches eine Schnittstelle zwischen einer Software und einem Debugger herstellt. Dieses Protokoll definiert Json Strukturen, damit Breakpoints, Ereignisse und weitere Debug Befehle an den Debugger gesendet werden können. Der Debugger führt dann diese Befehle an dem zu debuggenden Prozess – in diesem Fall der Prozess, der die Abläufe ausführt (z.B. Coreflow.Host) – aus.

Um einen Breakpoint setzen zu können muss der Inhalt der Quellcodedatei bekannt sein, damit der Debugger den Breakpoint einer Codezeile zuordnen kann. Da der Benutzer einen Breakpoint auf einen Code-Creator in der Administrationsoberfläche setzt, muss der Code-Creator zuerst einer Quellcodezeile zugeordnet werden. Der Austausch des tatsächlich generierten Quellcodes findet übet die Coreflow Api statt (siehe 7.4).

Wie in Abbildung 5 dargestellt, muss sich mindestens netcoredbg auf dem System befinden, welches untersucht werden soll. Der Debugger muss nicht installiert werden und kann somit direkt vom Coreflow.Host gestartet werden.

In dieser Arbeit ist das Debugging so implementiert, dass sich alle Komponenten auf dem gleichen System befinden müssen. Doch es ist theoretisch möglich, dass ein Ablauf Remote über Netzwerk untersucht werden kann. Um dies zu realisieren müsste ein Proxy implementiert werden, der die DAP Befehle vom Debugger übers Netzwerk transportiert. Alternativ kann ein netzwerkfähiger Debugger verwendet werden. Tatsächlich unterstützt der in dieser Arbeit genutzte Debugger „netcoredbg“ eine TCP Verbindung, doch hat diese bei einem Test nicht zuverlässig funktioniert.

Außerdem ist es möglich einen Ablauf über Visual Studio zu debuggen. Dafür wurde ein Code-Creator implementiert, mit diesem Code erzeugt wird, der einen installierten Just-In-Time Debugger startet. In dieser Arbeit wird aber angenommen, dass in der Praxis auf einer Produktiv- bzw. Testumgebung kein Visual Studio bzw. ein Just-In-Time Debugger installiert ist.

7.4 Coreflow Api

Mittels dieser Schnittstelle ist es möglich das Ergebnis der letzten Ablauf-Kompilierung der Coreflow Bibliothek abzufragen. Diese Daten werden benötigt, damit Coreflow.Web die Quellcodezeilen eines Breakpoints bestimmen kann. Die Coreflow Api verfügt derzeit nur über diese Funktion, könnte aber falls benötigt in Zukunft um weitere Funktionen ausgebaut werden. Implementiert ist sie über eine TCP Verbindung, die Json Strukturen austauscht.

7.5 Ablauf Validierung

Während der Entwicklungsphase eines Ablaufs kommt es vor, dass ein Code-Creator verändert wird. Falls die Parameter eines Code-Creator verändert werden, sind alle existierenden Abläufe, die diesen Code-Creator verwenden, nicht mehr konsistent zu der Implementierung der Code-Creator. Die Abläufe verwenden nun Parameter, die es ggf. nicht mehr gibt oder verwenden zu wenige Parameter. Um diesem Problem entgegenzuwirken überprüft Coreflow die Konsistenz eines Ablaufs. Im Namespace Coreflow.Validation.Checker sind Klassen implementiert, die jeweils überprüfen, ob eine bestimmte Eigenschaft erfüllt ist. Zum Beispiel überprüft der ParameterButNoArgumentChecker ob es Parameter ohne Argumente gibt. Wenn dies der Fall ist, wird eine Fehlermeldung erzeugt, die dann beim Öffnen des Ablaufs in Coreflow.Web angezeigt wird (siehe Abbildung 6).

Um die Fehler in einem Ablauf wieder beheben zu können sind Korrektoren implementiert, die Anhand der Fehlermeldungen erkennen, welche Korrektur möglich ist. Bei einem fehlenden Argument, kann z.B. der passende Korrektor ein neues leeres Argument erzeugen.

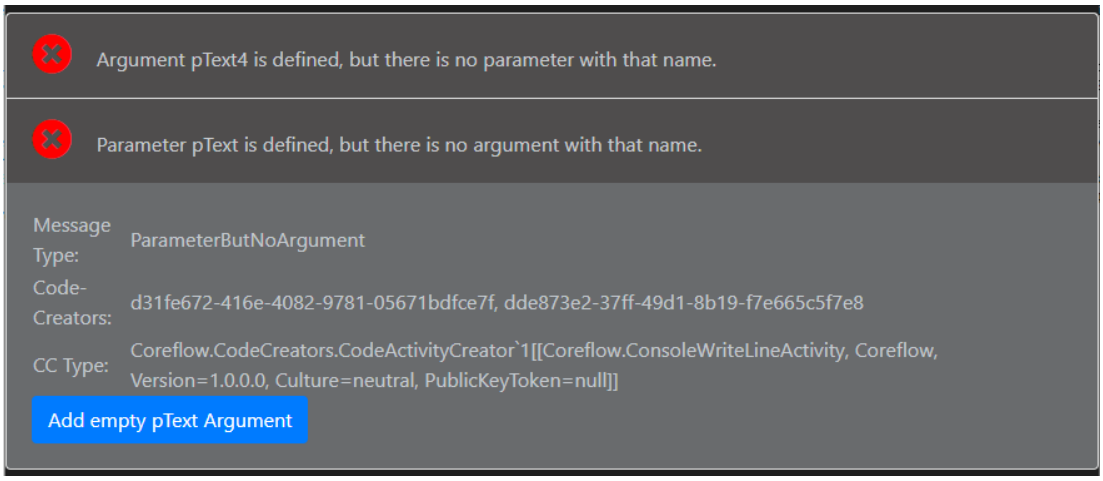

Abbildung 6: Ergebnis einer Validierung mit Fehler

7.6 Argument Injection

Ein Ablauf kann z.B. Daten enthalten, die nicht in einem serialisierten Ablauf oder auf einer Flow Repository gespeichert werden sollten. Dabei handelt es sich z.B. um Passwörter oder systemspezifische Konfigurationen.

Um diesem Problem entgegenzuwirken ist es möglich Variablen als Argument zu übergeben. Bei einer Verwendung der Argument Injection wird vor dem Aufruf automatisch Code generiert, der den tatsächlichen Wert der Variable abruft und diesen dann im Ablauf verwendet. Eine Implementierung dieser Argument Injection benutzt z.B. eine Argument.json Datei, die ein Key-Value Wörterbuch beinhaltet.

Ein Variablenname für die Argument Injection wird bei Coreflow immer mit einem $ Zeichen als Präfix repräsentiert.

Bei einer Json Datei mit folgendem Inhalt kann die Variable $EmailPassword in einem Ablauf verwendet werden, um den Wert „geheim“ abzurufen:

```
{
  "EmailPassword": "geheim"
}
```

7.7 Administrationsoberfläche: Kommentare und Farben

Um einem Administrator das Erstellen und Modifizieren eines Ablaufs zu erleichtern bzw. dies zu unterstützen kann ein Code-Creator mit einer Notiz und einer Farbe versehen werden. Zum Beispiel können alle noch nicht fertig implementierten Code-Creator markiert werden. Durch die Notizen können Administratoren Kommentare zur Dokumentation eines Ablaufs ergänzen.

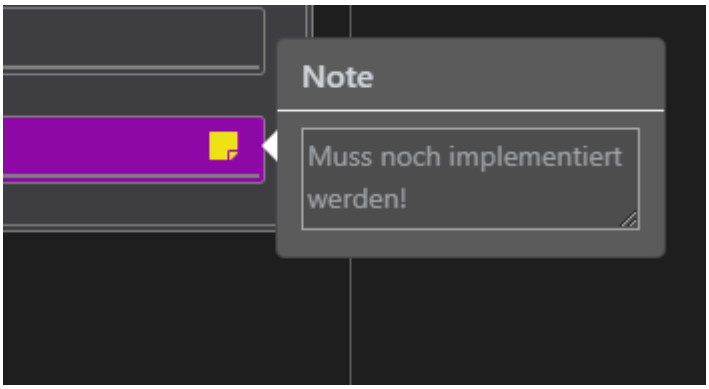

Abbildung 7: Code-Creator mit Notiz und Farbe

8 Ergebnis

Während dieser Arbeit wurde eine Ablaufsteuerung erstellt, die fast alle Anforderungen erfüllt. Die Ablaufsteuerung kann in eine LCP integriert werden, um damit dann eine Software erstellen zu können. Durch die in Kapitel 7 implementierten Werkzeuge, kann die Ablaufsteuerung aber auch ohne eine komplette LCP eingesetzt werden.

Ein Ablauf kann über die implementierte Weboberfläche erstellt und modifiziert werden, über ein Repository mit mehreren Hosts synchronisiert und dann ausgeführt werden. Er lässt sich In- bzw. Exportieren und kann dadurch nicht nur über die Repository ausgetauscht werden. Die Software ist plattformunabhängig und wurde auf Windows und Linux getestet. Über einen Debugger kann ein Ablauf angehalten und gestartet werden.

Die Anforderung, dass ein Ablauf verarbeitet werden kann, wenn unbekannte Datentypen referenziert werden bzw. Platzhalter für fehlende Code-Creator erzeugt werden, konnte nicht erfüllt werden. Der Grund dafür ist, dass die Deserialisierung des Ablaufs in so einem Fall fehlschlägt. Für inkonsistente Parameter gilt dieses Verhalten aber nicht. Diese Fehler können durch die Ablauf Validierung und Korrektoren behoben werden (siehe 7.5).

Folgende Grafik stellt eine Beispielkonfiguration der Arbeit dar:

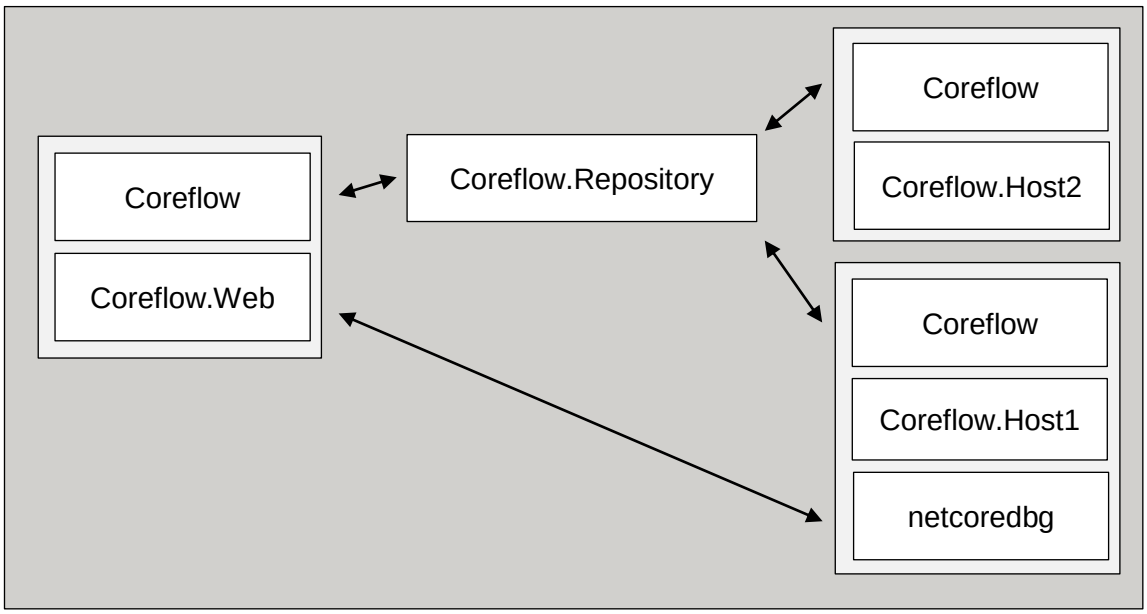

Abbildung 8: Beispielkonfiguration

8.1 Service Monitoring System

Dieses implementierte Anwendungsbeispiel soll demonstrieren, wie aus wenigen einzelnen Bausteinen ein Ablauf erstellt werden kann, der eine praktische Aufgabe löst. Die Aufgabe dieses Service Monitoring Systems ist nicht wie bei vielen anderen Monitoring Systemen das Überwachen eines Ports, sondern es sollen Dienste auf mehreren Servern überwacht werden.

In diesem Szenario wird ein Apache2 Webserver auf zwei Linux Servern überwacht. Es gibt einen Ablauf, der alle 10 Minuten den Zustand der beiden Dienste überprüft und das Ergebnis der Überprüfung in einer Datenbank abspeichert. Zusätzlich werden die Logdateien der Fehler abgespeichert. Somit werden auch Probleme registriert, die zwar nicht zum Absturz der Software geführt haben, trotzdem aber relevant für den korrekten Betrieb sein könnten.

Tabelle 1 stellt die Tabelle der Datenbank mit Beispieldaten dar.

server	servicename	online	message	time
1	Apache2	1	[…]	2019-06-12 12:14:34
1	Apache2	0	[…]	2019-06-12 12:24:34
1	Apache2	1	[…]	2019-06-12 12:34:34

Tabelle 1: Beispielwerte Service Monitoring Datenbank

In der Spalte message wird der Inhalt der Logdatei abgespeichert. Aus Darstellungsgründen wurde dieser Text in der oben gezeigten Tabelle entfernt.

Mit der Open Source Software Grafana werden die Daten in der Tabelle ausgelesen und dann visualisiert. Ein Administrator kann somit den Zustand der Dienste auf mehreren Servern in einer Übersicht überprüfen.

In Abbildung 9 ist eine solche Übersicht dargestellt. In einem Diagramm wird der Zustand des Apache Webservers der beiden Server 1 und 2 dargestellt. Zusätzlich werden die neusten Logeinträge der beiden Server angezeigt.

Abbildung 9: Visualisierung mit Grafana

In diesem implementierten Anwendungsfall wurde mit Coreflow eine Middleware zwischen dem Apache Webserver und Grafana erstellt. Coreflow übernimmt nur einen Teil des Gesamtkonzepts, doch nehmen die Abläufe, die diese beiden Softwarekomponenten miteinander verknüpfen, eine zentrale Rolle in dem System ein.

Die Realisierung mit Coreflow hat nun den Vorteil, dass, falls sich z.B. das Verhalten vom Apache2 durch ein Update verändert, nur der Ablauf angepasst werden muss. Die Middleware muss nicht in einer IDE neu kompiliert und verändert werden. Auch ist für die Anpassung ggf. kein Programmierer nötig, da der Administrator diese selbst vornehmen kann. Zusätzlich können die erstellten Activities für andere Dienste verwendet werden.

8.2 Performance

Die Performance eines Ablaufs hängt logischerweise von seiner Implementierung ab. Somit kann nicht allgemein bestimmt werden, wie lange es dauert einen Ablauf auszuführen. Die Laufzeit ist auch von der Hardware und dem System abhängig. Dieses Kapitel soll aber eine grobe Übersicht über die Performance von Coreflow darstellen.

Getestet wurde auf Windows 10 x64 mit einem AMD Ryzen Threadripper 1920X CPU mit 3.6 Ghz und DDR4 RAM mit 2134Mhz.

Die Software ist eine .NET Core 3 preview 4 Konsolenanwendung, welche mit Debug Symbolen kompiliert wurde und ohne Debugger ausgeführt wird.

Starten und beenden eines leeren Ablaufs ohne Speicherung der Instanzen:

Aufgabe	**Gesamte Laufzeit**
Laden und Kompilieren des Ablaufs	4000ms
Erster Aufruf	2ms
Zweiter Aufruf	0.1ms
10 Aufrufe	0.02ms
100 Aufrufe	0.08ms
100.000 Aufrufe	75ms

Tabelle 2: Performance: Aufruf von Abläufen

In Tabelle 2 sind die Messwerte eines Performancetests dargestellt. Zu sehen ist, dass das Laden, Quellcodeerzeugung und die Kompilierung der Abläufe ein Vielfaches der Durchlaufzeit eines leeren Ablaufs entsprechen. Dies liegt daran, der der Roslyn Compiler bei erstem Aufruf viele .dll Dateien und dessen Datentypen etc. laden muss. Diese Informationen werden zwischengespeichert, sodass ein weiterer Kompiliervorgang dann schneller ausgeführt werden kann. Ein Ablauf muss aber nur einmal vor dem ersten Aufruf und bei Änderung kompiliert werden.

Die Laufzeit der Aufrufe ist ab 100.000 linear. Doch bei weniger Aufrufen wirken sich CPU Caches und Optimierungen, sowie der Scheduler des Betriebssystems auf die Laufzeit der Aufrufe aus. Beim ersten Aufruf eines Ablaufes

ist dieses Verhalten sehr gut zu beobachten. Der erste Aufruf ist ca. 20x langsamer ist als der zweite Aufruf. Bei 100 Aufrufen ist ein Aufruf 2.5x schneller als ein einzelner Aufruf.

Ein weiterer Performanceverlust entsteht durch die Erzeugung eines Objekts für eine Activity. Die Klasse der Activity wird in einem Ablauf immer neu instanziiert und bei einem Aufruf wird die „Execute“ Methode des Objekts aufgerufen. Coreflow optimiert aber die Erzeugung dieser Objekte, sodass, falls z.B. zweimal die „ConsoleWriteLineActivity“ verwendet wird, das Objekt nur einmal erzeugt wird. Die Erzeugung des Objekts erfolgt auch immer vor einer Schleife, sodass auch bei Schleifen das Objekt (meistens) nur einmal erstellt wird. Aufgrund dieses Verhaltens lässt sich kein aussagekräftiger allgemein gültiger Performancetest erstellen.

In der Praxis ist diese Erzeugung sowieso kaum messbar, da die Implementierungen der Activities eine längere Laufzeit haben oder sogar Datei oder Netzwerkaufrufe getätigt werden. Durch die Struktur eines Ablaufs werden ggf. auch bestimmte Objekte nicht erzeugt, da der Code aufgrund der Argumente nie erreicht wird.

Zusammengefasst kann der grobe Schätzwert von 1ms Overhead pro Aufruf angenommen werden.

8.3 Bewertung des Konzepts

Das Konzept definiert die Grundlagen zur Implementierung einer Ablaufsteuerung. Es beschreibt nur die Kernkomponenten, die benötigt werden, um eine Ablaufsteuerung zu realisieren. Die Implementierung (siehe Kapitel 6) stellt dar, dass sich dieses Konzept umsetzen lässt. Durch die Definition des UI- und Ablauf-Modells ist die Komplexität der Implementierung der beiden Projekte noch überschaubar. Das Konzept definiert keine Anforderungen an die Performance, doch es wird angenommen, dass bei korrekter Implementierung die Anforderung, dass ein Benutzer keinen Performanceverlust war nimmt, erfüllt ist. Durch die im Konzept definierten Aktivities ist für einen Programmierer ein einfacher Einstieg möglich, ohne genaue Details über einen Code-Creator verstehen zu müssen. Für einen Administrator sollte es durch die einfache Bedienung mit Drag and Drop nach kurzer Einarbeitung möglich sein, vorhandene Abläufe zu verstehen und diese zu modifizieren.

Die im Konzept definierte Trennung der Bibliothek und der Administrationsoberfläche ermöglicht die Integration der Bibliothek in ein anderes Produkt, welches mit .NET Core implementiert bzw. kompatibel ist. So können Abläufe auch ohne die Administrationsoberfläche verwendet werden.

Im Konzept werden die in Kapitel 7 implementierten Werkzeuge nicht betrachtet. Sie sind optionale Erweiterungen und haben nur indirekt mit dem Ansatz, Quellcode aus Code-Creator zu generieren, zu tun.

8.4 Coreflow und Microservices

In Kapitel 3.2 wird eine Diskussion über Microservices mit Workflow-Engines dargestellt. Aber wie ist jetzt Coreflow einzuordnen?

Meiner Meinung nach kann Coreflow als „Workflow-Engine“ für einen Microservice eingesetzt werden. Dabei ist aber zu beachten, dass Coreflow kein BPMN Interpreter ist und somit Arbeitsabläufe nicht direkt modelliert werden können. Coreflow ist nach dieser Definition keine Workflow-Engine, welche Arbeitsabläufe ausführen kann. Im Blogbeitrag „Wer Microservices richtig macht, braucht keine Workflow Engine und kein BPMN“ [8] wird z.B. der „BestellEingangService“ dargestellt: „Wird synchron vom Kunden aufgerufen und hat im Wesentlichen die Aufgabe, eine eindeutige ID der Bestellung zu erzeugen“. Dieser Service kann nun entweder klassisch im Code implementiert oder z.B. mit Coreflow realisiert werden. Die einzelne Aufgabe des Microservices ist nämlich eigentlich ein technischer Schritt: Erstellung einer eindeutigen ID. Die Zusammenhänge der Microservices ergeben dann aber den Arbeitsablauf.

Ein komplett anderer Ansatz wäre auf eine Microservice Architektur im Sinne von REST Schnittstellen und Docker Containern zu verzichten. Einzelne Aufgaben können durch verschiedene Abläufe modelliert werden, die sich dann gegenseitig aufrufen oder über einen lokalen Eventbus kommunizieren. Dabei würde die Software wieder auf einer Hardware laufen, doch durch Coreflow gibt es keine feste Kopplung der einzelnen Softwarekomponenten. Der Vorteil von Microservices, dass je ein Team einen bestimmten Service entwickeln kann, kann auch mit Coreflow umgesetzt werden. Die Abläufe, sowie einzelne Softwarekomponenten können von mehreren Teams entwickelt und dann zusammengesetzt werden. Natürlich skaliert die Umsetzung mit Coreflow auf einem lokalen System nicht und eine Ausfallsicherheit muss über z.B. eine externe Datenbank realisiert werden, doch sind Probleme mit dem Eventbus oder mit der erhöhten Netzwerklast durch einzelne Microservices bei so einer Lösung ausgeschlossen. Durch die Code Generierung der Abläufe ist der Overhead in Ausführung gering und somit kann ein System auch eine gewisse Menge an Anfragen, Kunden oder Mitarbeiter bewältigen – Vielleicht ist eine komplexe Microservice Architektur für gar nicht nötig?

9 Ausblick

Diese Ablaufsteuerung ist ein Projekt, welches vermutlich immer erweitert und verbessert werden kann. Doch sollte zuerst die Implementierung der Deserialisierung verbessert werden, damit auch bei nicht mehr vorhandenen Datentypen ein Ablauf deserialisiert werden kann.

Eine Erweiterung wäre die Verbesserung der Darstellung der Instanzen, sowie ein Metadatenspeicher für eine Instanz. So könnten entstehende Daten einer Instanz zugeordnet gespeichert werden, die dann auf der Administrationsoberfläche angezeigt werden. Zusätzlich könnten Logeinträge in einem Metadatenspeicher abgelegt werden, sodass eine Instanz besser nachvollzogen werden kann. Auch wäre es möglich, alle aufgerufenen Code-Creator in Aufrufreihenfolge und ggf. dessen Argumenten abzuspeichern. Somit kann ein modellierter Ablauf besser verstanden und analysiert werden.

Die Unterstützung von macOS ist theoretisch möglich, doch wurde in dieser Arbeit keinerlei Tests durchgeführt, ob dies auch der Praxis entspricht. Bezüglich der Anforderung für die Plattformunabhängigkeit sollte dies überprüft und ggf. verbessert werden.

Um eine Dokumentation der Code-Creator und dessen Parameter zu ermöglichen, sollte es möglich sein, Beschreibungen für die Parameter, Rückgabewerte und dem Code-Creator selbst im Quellcode definieren zu können. Diese Texte sollten dann in der Administrationsoberfläche angezeigt werden.

Da Variablen meistens automatisch erstellt werden, sollte dies in der Administrationsoberfläche visualisiert werden. Falls ein Administrator einen Variablennamen eingibt und annimmt, dass bereits eine vorhandene Variable verwendet wird, aber eine automatische Erstellung visualisiert wird, kann er einfacher feststellen, dass z.B. der Variablenname falsch eingegeben wurde.

Literaturverzeichnis

[1] "Low-Code-Plattform" Wikipedia [Online]. Available: https://de.wikipedia.org/wiki/Low-Code-Plattform. [Accessed 2 April 2019].

[2] "Computerwoche: Low-Code-Plattformen auf einen Blick" [Online]. Available: https://www.computerwoche.de/a/low-code-plattformen-auf-einen-blick,3544905. [Accessed 2 April 2019].

[3] V. A. Ramalingam, Introducing Microsoft Flow, New Jersey, USA: Apress, 2018, pp. 1-10.

[4] T. S. Vlad Catrinescu, Deploying SharePoint 2019 Installing, Configuring, and Optimizing for On-Premises and Hybrid Scenarios, Apress, 2019.

[5] F. Consulting, "The Total Economic Impact Of PowerApps And Microsoft Flow" 2018.

[6] A. Schwartz, "Microservices Mehr als nur ein Hype?" *Informatik-Spektrum,* vol. 40, no. 6, p. 590–594, Dezember 2017.

[7] D. M. Bernd Rücker, "Wie lässt sich Ordnung in einen Haufen (Micro-) Services bringen?" 21 August 2015. [Online]. Available: https://jaxenter.de/wie-laesst-sich-ordnung-in-einen-haufen-micro-services-bringen-23938. [Accessed 13 06 2019].

[8] T. Flore, "Wer Microservices richtig macht, braucht keine Workflow Engine und kein BPMN" codecentric Blog, 1 09 2015. [Online]. Available: https://blog.codecentric.de/2015/09/wer-microservices-richtig-macht-braucht-keine-workflow-engine-und-kein-bpmn/. [Accessed 13 06 2019].

[9] C. Richardson and J. R. Rymer, "The Forrester Wave™: Low-Code Development" *The Forrester Wave™,* vol. 1, p. 18, 2016.

[10] Microsoft, "Introduction to ASP.NET Core" [Online]. Available: https://docs.microsoft.com/de-de/aspnet/core/. [Accessed 29 Mai 2019].

[11] F. Reynders, in *Modern API Design with ASP.NET Core 2*, Odijk, Apress, 2018, pp. 9-12.

[12] M. Withopf, "Scheibenweise" *iX,* vol. 04, 2012.

[13] N. Harrison, Code Generation with Roslyn, Lexington, South Carolina, USA: Apress, 2017.

[14] "Github LiteDB" [Online]. Available: https://github.com/mbdavid/LiteDB. [Accessed 30 Mai 2019].

[15] "Github Monaco Editor" [Online]. Available: https://github.com/microsoft/monaco-editor. [Accessed 21 Mai 2019].

[16] "Github ExtendedXmlSerializer" [Online]. Available: https://github.com/wojtpl2/ExtendedXmlSerializer. [Accessed 9 April 2019].

[17] "Github Roslyn" [Online]. Available: https://github.com/dotnet/roslyn/wiki/Runtime-code-generation-using-Roslyn-compilations-in-.NET-Core-App. [Accessed 05 Juni 2019].

[18] "MICROSOFT PRE-RELEASE SOFTWARE LICENSE TERMS" [Online]. Available: https://visualstudio.microsoft.com/license-terms/mt644895/. [Accessed 14 Mai 2019].

[19] "Github netcoredbg" [Online]. Available: https://github.com/Samsung/netcoredbg. [Accessed 14 Mai 2019].

[20] "Blog Jetbrains" [Online]. Available: https://blog.jetbrains.com/dotnet/2017/02/15/rider-eap-17-nuget-unit-testing-build-debugging/. [Accessed 14 Mai 2019].

[21] "Github .NET Core" [Online]. Available: https://github.com/dotnet/core/issues/505. [Accessed 14 Mai 2019].

Anhang A. Debugger für .NET Core

In den „MICROSOFT PRE-RELEASE SOFTWARE LICENSE TERMS” der “MICROSOFT .NET CORE DEBUGGER COMPONENTS” wird mit dem Satz “You may only use the .NET Core Debugger Components with Visual Studio Code, Visual Studio or Xamarin Studio software to help you develop and test your applications.” [18] die Benutzung des Debuggers für eine, wie in dieser Arbeit erstellten Plattform, verboten. Aber auch andere Firmen, die Entwicklungsumgebungen entwickeln, sind von diesem Problem betroffen. Jetbrains hat Anfang 2017 in ihrem Blog berichtet, dass ihre IDE das Debuggen von .NET Core nicht unterstützt [20]. Auf dem Github Repository von .NET Core gibt es dementsprechend eine Diskussion über dieses Thema. Die aktuelle Meinung auf Github ist, dass Microsoft weiterhin Geld mit Visual Studio verdienen will und somit den Debugger nie veröffentlichen wird. Dies widerspricht leider dem Open Source Grundgedanken von .Net Core [21].

Aufgrund dieser Einschränkung verwendet diese Arbeit den von Samsung implementierten Open Source .Net Core Debugger netcoredbg [19]. Samsung hat diesen Debugger entwickelt, da es mit Tizen .NET möglich ist, Apps für das von Samsung entwickelte Betriebssystem Tizen zu entwickeln.

Anhang B. Monatsberichte

01.03.19 – 28.03.19

Durchgeführte Arbeiten

Nachdem ich ein grobes Konzept ausgearbeitet habe, implementierte ich zuerst einen Workflow Core. Also eine Bibliothek, die aus einem XML C# Code erstellen kann, der dann kompiliert wird. Dazu wurde ein trivialer Workflow für einen Unittest erstellt, aus den zuerst nur C# Code erstellt wird. Iterativ ergänzte ich die Implementierung mit der XML Serialisierung und dem Kompilieren, sodass das Proof of Concept für den Core grundlegend funktionierte.

Im nächsten Schritt startete ich die Implementierung von einer ASP.Net Core Anwendung, die für den Workflowersteller die benötigte Oberfläche zur Verfügung stellt. Nach und nach wurden einzelne Funktionen hinzugefügt, die es ermöglichen den Workflow zu bearbeiten. Durch die Oberfläche konnte ich einfacher neue Workflows erstellen, die ich zum Testen des Cores verwenden konnte. Dabei konnte ich einige Bugs im Core finden, die ich dann behoben habe.

Für die Thesis habe ich mir eine erste Struktur ausgedacht und diese mit ersten Stichworten über den Inhalt des jeweiligen Kapitels versehen.

Erzielte Ergebnisse

Ein Workflow kann in ein XML serialisiert und deserialisiert werden. Aus einem XML lässt sich meistens kompilierfähiger C# Code erstellen, der dann ausgeführt werden kann. Somit lassen sich Workflows derzeit im Dateisystem abspeichern und wieder einlesen.

Das Admintool kann manche Workflow Informationen wie Argumente und Namespaces darstellen, doch Variablen werden derzeit noch nicht unterstützt. Auch lassen sich die Werte noch nicht bearbeiten.

Es gibt an einigen Stellen noch hardgecodete Annahmen und Werte.

Die Thesis hat nun eine grobe Struktur.

Abweichungen / Probleme

Leider hat sich herausgestellt, dass derzeit noch keine if-else Activity umsetzen lässt, da ich dies beim Design des Cores nicht bedacht habe. – Diesem Problem werde ich mich nächsten Monat widmen.

Es war nicht einfach einen geeigneten Serializer zu finden, der in der Lage ist Interfaces korrekt zu (de-)serialisieren. Auch hat sich erst während der Entwicklung herausgestellt, dass es Sinn macht, eine zweite Objektstruktur eines Workflows für die UI zu machen, da diese sich von der Workflowstruktur im XML unterscheidet.

Auch der Ansatz für die Konvertierung von XML zu C# musste geändert werden, da es nötig ist Zwischenschritte zu kompilieren, damit die definierten Variablen bestimmt werden können.

Ausblick über die geplanten Tätigkeiten und Ergebnisse des nächsten Berichtszeitraums

Nächsten Monat will ich mich hauptsächlich auf das Thesis Dokument konzentrieren, da ich nun weiß, dass mein Ansatz grundlegend funktioniert.

Derzeit ist mir noch nicht klar wo die Grenzen dieser Implementierung sind – Ist es möglich einen Workflow anzuhalten und mit aktuellen Variablen zu serialisieren?

01.04.19 – 30.04.19

Durchgeführte Arbeiten

Diesen Monat habe ich das Aufrufen eines anderen Flow implementiert, dazu .NET Core aktualisiert und Anpassungen an der Erstellung der Assemblies gemacht. Auch werden nun die Variablentypen der Parameter erkannt und können ggf. automatisch konvertiert werden. Zusätzlich erstellte ich den ersten Flow, der in einer Produktivumgebung die Auslastung der Festplatte überprüft und gegebenenfalls eine Email versendet. Dabei gab es beim ersten Langzeittest keinerlei Probleme und das System arbeitet derzeit seit mehreren Tagen ohne Zwischenfälle oder Fehler.

Eine Testumgebung wurde auch das erste Mal auf einem Linux Server installiert und auf Funktionalität überprüft. Zu meinem Thesis Dokument habe ich eine ausführliche Beschreibung einer Konvertierung vom Ablauf-Modell zum Quellcode formuliert, sowie einige Teile für die Einführung geschrieben.

Auch wurden einige Definitionen und Begrifflichkeiten verbessert, sodass das Dokument nun diese konsistent verwendet.

Erzielte Ergebnisse

Ein Flow kann nun einen anderen Flow aufrufen und Parameter an diesen Flow übergeben. Es ist somit nun möglich, dass Flows untereinander verschachtelt werden können.

Die Datentypen der Code-Creator Parameter werden nun über Roslyn erkannt und es ist nun möglich diese auch automatisch konvertieren zu können. Wenn also z.B. ein Parameter den Typ String IEnumerable hat, der angegebene Parameter aber nur ein einfacher String ist, kann der String automatisch in ein Array<string> konvertiert werden.

Das Thesis Dokument hat nun mehr Inhalt und enthält eine Beschreibung der Konvertierung vom Ablauf-Modell zum Quellcode.

Das Problem vom vorherigen Monat, dass ein if-else Code-Creator nicht realisierbar ist wurde behoben.

Abweichungen / Probleme

Damit ein Flow einen anderen Flow aufrufen kann, muss dieser den Datentyp des Flows kennen. Somit werden nun alle Flows in einem Assembly erzeugt. Das in .NET Core 3 Preview 4 neue Feature um Assemblies entladen zu können funktioniert manchmal, sorgt aber auch ggf. zu Komplettabstürzen der .NET Runtime. Derzeit wurde das Entladen von Assemblies somit wieder deaktiviert – soll aber in Zukunft wieder genutzt werden um Memory Leaks bei Veränderung eines Flows zu verhindern.

Ein Flow kann nicht deserialisiert werden, wenn Drittanbieter Bibliotheken fehlen. Dies sollte laut Anforderungen möglich sein, doch gibt es derzeit noch keine Lösung für dieses Problem.

01.05.19 – 31.05.19

Durchgeführte Arbeiten

Das Thesisdokument wurde in diesem Monat um das Kapitel weitere Werkzeuge ergänzt und ich habe am Debugger und an der Validierung entwickelt. Nach Problemen mit der Lizensierung des Microsoft Debuggers – welcher auch nicht Open Source ist - habe ich eine Alternative gefunden und diese dann implementiert. Dabei musste die Roslyn Kompilierung etwas angepasst werden und die CoreflowApi wurde hinzugefügt, um den Quellcode der generierten Abläufe austauschen zu können. Ich habe auch erste Activities für das Service Monitoring gebaut und einen Ablauf entwickelt, doch funktioniert das Anwendungsbeispiel noch nicht zuverlässig.

Die Administrationsoberfläche wurde um Kommentare und Farben erweitert. Auch wurde die Argument Injection über ein Dictionary und über eine Json Datei implementiert.

Das Logging wurde über „Microsoft.Extensions.Logging“ implementiert, sodass verschiedene Logbibliotheken verwendet werden können.

Einige Platzhaltergrafiken wurden im Dokument ausgetauscht, sowie kleinere Änderungen der Kapitel durchgeführt.

Erzielte Ergebnisse

Das Dokument enthält nun fast alle Kapitel und hat fast den geforderten Umfang.

Mit dem Debugger kann ein Ablauf über die Weboberfläche angehalten und wieder gestartet werden. – Derzeit wird aber die aktuelle Position des Programmzeigers nicht angezeigt. Breakpoints können über die Weboberfläche gesetzt und gelöscht werden. Der Quellcode für den Debugger wird über die CoreflowApi ausgetauscht.

Die Argument Injection funktioniert und es können Argumente in einer Json Datei angegeben werden. Die Coreflow Bibliothek kann mit verschiedenen Logbibliotheken genutzt werden.

Eine Activity kann über die Administrationsoberfläche mit Kommentaren und Farben versehen werden.

Abweichungen / Probleme

Das Problem vom letzten Monat, dass ein Ablauf nicht deserialisert werden kann, falls ein referenzierter Typ nicht vorhanden ist, wurde nicht behoben. Aufgrund der Zeitplanung und anderen Prioritäten wird dieses Feature nicht mehr im Rahmen der Bachelorarbeit implementiert.

Die Implementierung des Debuggers hat viel Zeit in Anspruch genommen, da es keinen Open Source Debugger von Microsoft gibt. Das Service Monitoring System bzw. die Coreflow Installation auf einem Testserver funktioniert nicht zuverlässig. Es werden nicht reproduzierbare Fehler beim lesen von Dateien des .NET Core SDKs gemeldet. – Es liegt vermutlich ein defekt der Server Festplatte vor.

01.06.19 – 23.06.19

Durchgeführte Arbeiten

Der Fokus lag während diesem Berichtszeitraum bei dem Thesisdokument. Eigentlich hat Anfang dieses Monates das Dokument schon den geforderten Umfang, doch benötigen bekanntlich die letzten 10% die Hälfte der Zeit. Das Kapitel Ergebnis und Ausblick wurde ergänzt, Grafiken hinzugefügt und viele Fehler korrigiert.

Coreflow habe ich das erste Mal auf einem OrangePi installiert und getestet. Nach einigen Anpassungen im Code, welche die Performance verbessern, kommt Coreflow nun mit den 2GB RAM vom Einplatinencomputer zurecht. Nachdem ich überprüft habe, wie sich ein kompilierter Ablauf referenzieren lassen kann, musste ich feststellen, dass dies noch nicht funktioniert. Nach einer Recherche stellt sich heraus, dass Coreflow gegen die falschen Assemblies kompiliert. Somit wurde ein Mapper implementiert, der meistens die passenden Assemblies finden kann.

Eine weitere Idee war, Coreflow auch mit dem „großen“ .NET Framework kompatibel zu machen. Bis auf wenige Zeilen Code war der Quellcode auch kompatibel. Nach den Anpassungen konnte Coreflow mit .NET Framework kompiliert werden.

Die Installation auf dem Debian Server wurde aktualisiert und es wurde eine Datenbank erstellt, die mit Grafana kommunizieren kann.

Erzielte Ergebnisse

Das Thesisdokument ist nun fertiggestellt. Es enthält die letzten fehlenden Kapitel, die fehlende Englische Übersetzung des Abstracts usw.

Coreflow wurde auf Debian x64 und ARM sowie Windows getestet und funktioniert. Der Apache Webserver wird von Service Monitoring System überwacht und Grafana visualisiert die gespeicherten Daten.

Durch das Caching der .dll Referenzen sind die zufällig auftretenden Fehler unter Linux behoben und eine Kompilierung dauert auf dem OrangePi nur noch wenige Sekunden.

Fehler in Coreflow.Web mit Parametern und Argumenten wurden behoben.

Es wurden einige Performancetests für Coreflow durchgeführt und dokumentiert.

Abweichungen / Probleme

Die letzten Arbeiten am Thesisdokument hatten etwas mehr Zeit gebraucht als angenommen. – Vermutlich resultiert dies aufgrund von Motivationsproblemen und vielen kleinen Anpassungen im Dokument.

Die erste Installation auf einem OrangePi zeigte, dass .NET Core zwar grundsätzlich plattformübergreifend funktioniert, doch es nicht automatisch heißt, dass Coreflow auch direkt auf allen Plattformen funktioniert. Zusätzlich hat die Installation bestätigt, dass es in Roslyn Funktionen gibt, die generell unter Linux nicht korrekt funktionieren. Somit wurde nun komplett auf diese Funktionen verzichtet.

Die Kompatibilität mit dem .NET Framework ist leider nicht möglich. Der Code lässt sich zwar kompilieren, jedoch sind einige Funktionen vom Roslyn Compiler nicht implementiert – Gegebenenfalls ist dies aber in der Zukunft möglich.