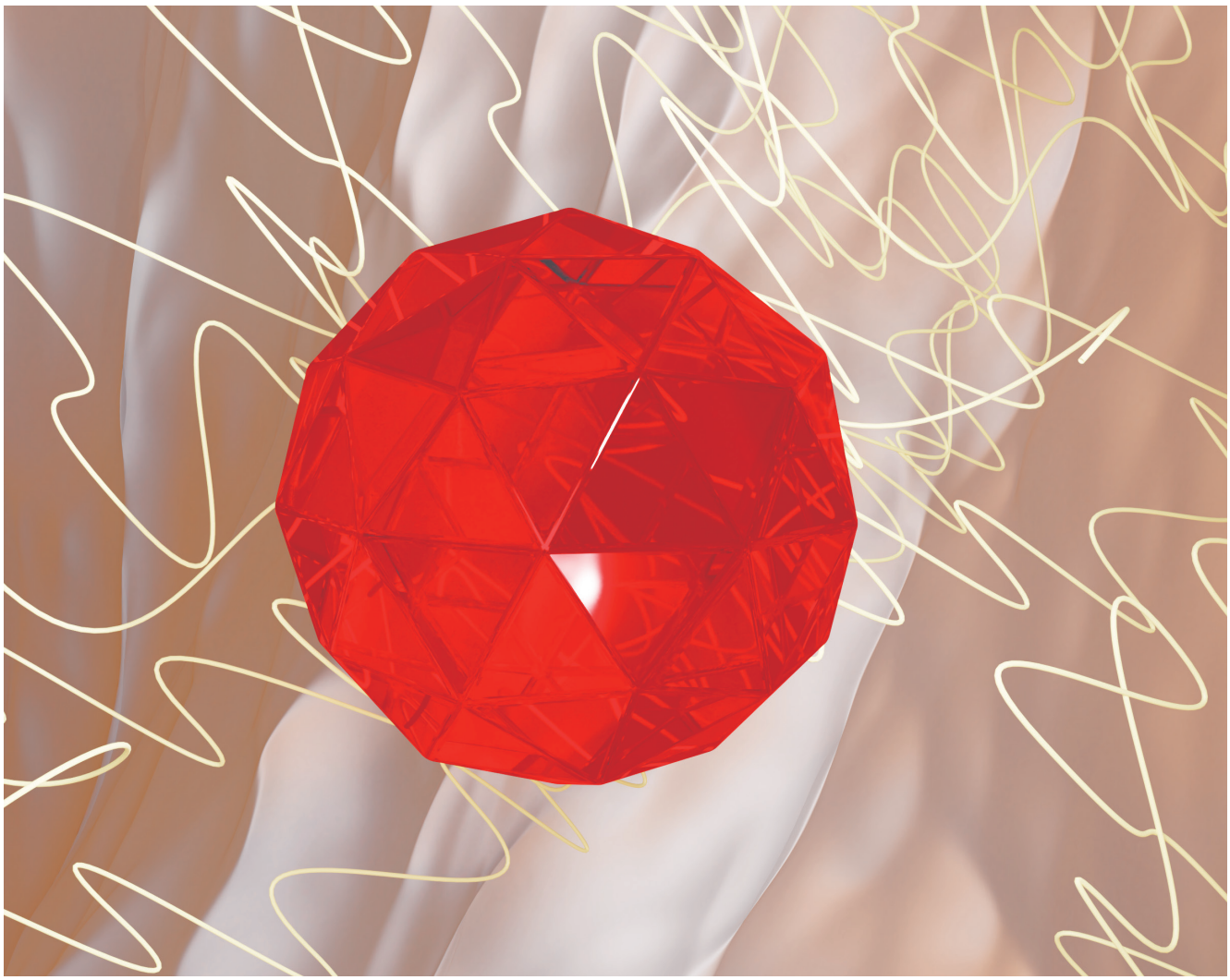


Bild: dotnetpro

OPEN-SOURCE-PROJEKTE FÜR .NET: ELSA WORKFLOWS

Andere arbeiten lassen

Mit Elsa lässt sich komplexe Geschäftslogik einfach visualisieren und bearbeiten.

Im .NET-Umfeld hat sich in den vergangenen Jahren eine gesunde Open-Source-Community entwickelt, die durchaus interessante und sinnvolle Projekte liefert. Leider werden diese angesichts der deutlich größeren und besser vermarkteten Open-Source-Lösungen aus dem Hause Microsoft häufig zu wenig beachtet. Dieser Artikel wirft einen genauen Blick auf das Projekt Elsa Workflows [1], kurz Elsa. Elsa ist ein Framework, um Arbeitsabläufe in .NET-Projekten zu bearbeiten und auszuführen. Dabei können diese Arbeitsabläufe – oder Workflows – durch Verknüpfung von Boxen in einem webbasierten Designer erstellt werden. Jede Box erhält Eingabeparameter und gibt als Ausgabe einen oder mehrere Werte zurück. Am Ende können die Workflows beliebig groß und komplex werden. Ebenso lassen sich Arbeitsabläufe mithilfe von C#-Code erstellen.

Worum geht's?

Beim Erweitern von Applikationen denken Entwickler häufig an Fabrikmuster oder Plug-ins – selten jedoch an die Automatisierung von Standardaufgaben über einen Workflow. Dabei ist Letzteres für Endanwender häufig wichtiger als das direkte Nutzen weiterer externer Funktionalitäten.

Ein Arbeitsablauf kann im Prinzip beliebig komplex aufgebaut werden. Die Grundsteine finden sich im Regelfall in der bereitstellenden Anwendung. So können Ereignisse, die von der Software ausgelöst werden, sinnvolle Trigger darstellen. Eingabeparameter oder Ausgaben sind häufig auch stark an die Applikationsdomäne gebunden. Wer es generischer möchte, kann einen HTTP-Aktuator als Ziel eintragen. Dadurch lassen sich Dienste wie Zapier oder IFTTT zur weiteren Bearbeitung ansprechen.

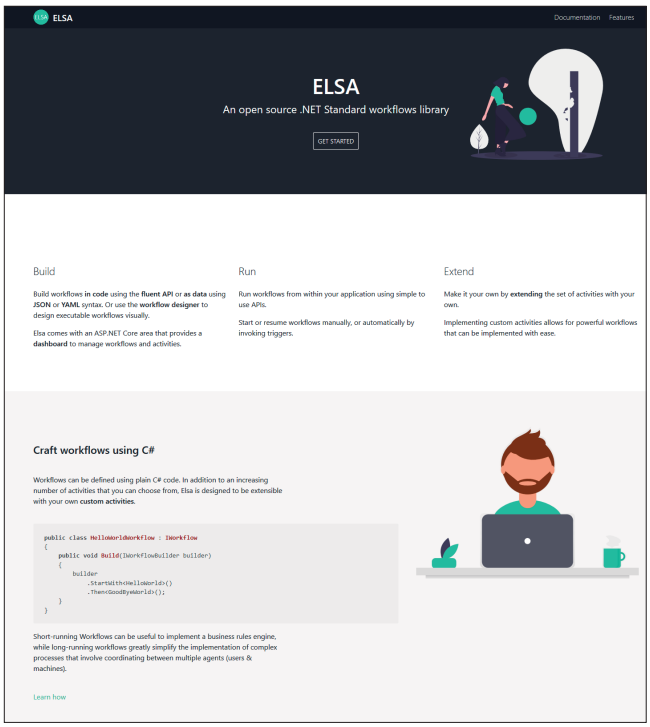
In der Praxis

Während Elsa in vielen Aspekten versucht, ein würdiger Nachfolger für die Workflow Foundation zu sein, so ist die allgemeine Tendenz, eine doch deutlich modernere Alternative zu repräsentieren. Das fängt bereits bei der sehr gut gestalteten, ausführlichen Dokumentation an und zieht sich über die Installation bis hin zur Konfiguration und Verwendung. Wie in Bild 1 dargestellt, ist über die Homepage des Projekts ohne große Umwege weiteres Lernmaterial zu erreichen.

Es gibt zwei Modelle, um die Arbeitspläne auszuführen:

- In-process: Arbeitspläne werden innerhalb der Anwendung angestoßen und ausgeführt.
- Out-of-process: Hier werden die Arbeitspläne in einem externen Prozess verwaltet, der Interaktionen über HTTP und AMQP zulässt. Somit kann jede beliebige Applikation die Workflow-Engine von Elsa verwenden – und zwar unabhängig vom benutzten Technologie-Stack. Dies erlaubt auch Zugriff von Skriptsprachen wie etwa PHP oder Ruby.

Elsa versteht sich hierbei als generische Workflow-Engine. Im Regelfall wird das Framework innerhalb einer ASP.NET-Anwendung (Core) eingebunden. In diesem Fall kann man bereits in den grundlegenden Diensten die Basis für Dependency Injection legen. Listing 1 zeigt die Integration von Elsa in einer ASP.NET-Core-Applikation.



Die Homepage von Elsa macht es dem Besucher einfach, an Informationen zu kommen (Bild 1)

Listing 1: Integration von Elsa in ASP.NET Core

```
public void ConfigureServices(
  IServiceCollection services)
{
  services
    .AddElsa()
    .AddTimerActivities();
}
```

Listing 2: Wiederkehrenden Workflow definieren

```
public class RecurringTaskWorkflow : IWorkflow
{
  public void Build(IWorkflowBuilder builder)
  {
    builder
      .AsSingleton()
      .StartWith<TimerEvent>(x =>
      x.TimeoutExpression = new
      LiteralExpression<TimeSpan>("00:00:05"))
      .Then<WriteLine>(x => x.TextExpression =
      new JavaScriptExpression<string>(
      "'It's now ${new Date()}. "
      + "Let's do this thing!'"));
  }
}
```

Zusätzlich ist es noch möglich, Elsa an dieser Stelle ausführlich zu konfigurieren. Zunächst sollte dafür das NuGet-Paket Elsa eingerichtet werden. Experten können anstelle der kombinierten Abhängigkeit auch mit Elsa.Core starten. Hierbei handelt es sich um Kernfunktionalitäten mit der geringsten Anzahl an Abhängigkeiten. Erweiterungen und Vorkonfigurationen wie zum Beispiel die Möglichkeit, *AddTimerActivities()* zu konfigurieren, sind hier noch nicht enthalten.

Die Arbeitspläne selbst sind vollständig serialisierbar. Dies führt dazu, dass diese nicht nur in C# erstellt werden, sondern auch aus JSON-, XML- und anderen Dateien importiert werden können. Auch das Speichern auf der Festplatte oder in einer Datenbank ist problemlos möglich.

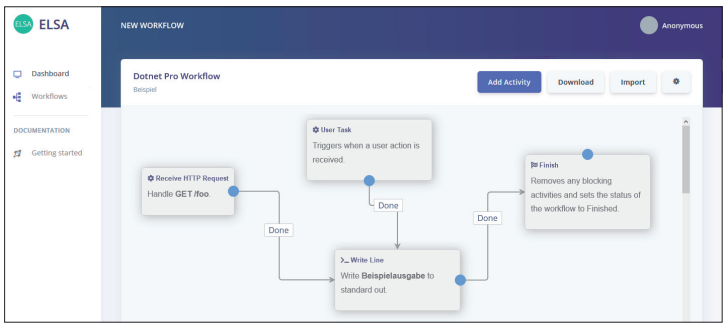
Für ein direktes und unaufwendiges Bereitstellen von Workflows genügt es, das *IWorkflow*-Interface zu implementieren. In der *Build()*-Methode lässt sich daraufhin der Workflow über das Fluent-API eines *IWorkflow*-Objekts erstellen. In Listing 2 wird darüber ein wiederkehrender Workflow erzeugt, der alle fünf Sekunden angestoßen wird. Über *JavaScriptExpression* ist es möglich, einen String zur Laufzeit dynamisch auszuwerten. Hier kann durchaus auch komplexere Logik verwendet werden.

Um zeitgesteuerte Aktivitäten zu ermöglichen, muss vorab der entsprechende Service hinzugefügt werden. In Listing 1 wurde der hierzu notwendige Code bereits eingefügt.

Der visuelle Designer wird über das NuGet-Paket Elsa. Dashboard zur Verfügung gestellt. Seine Installation ist genauso einfach wie bei Elsa selbst. Über die Erweiterungsmethode *AddElsaDashboard()* lassen sich die notwendigen ▶

Dienste konfigurieren. Zur optimalen Anzeige sollten statische Dateien vom ASP.NET-Core-Server ausgespielt werden. In Bild 2 ist die Standardansicht des Workflow-Editors dargestellt.

Besonders beeindruckend ist die direkt nach der Installation vorhandene Auswahl von Szenarien, Auslösern und Optionen. Das Framework erfüllt so ziemlich alle Wünsche bereits ohne irgendwelche weiteren Konfigurationen oder Installationen. Man kann also direkt loslegen, ohne viel Zeit mit der grundlegenden Integration und Bereitstellung verbringen zu müssen.



Erstellen eines Workflows mit Elsas visuellem Designer (Bild 2)

Einen Beitrag leisten

Im Vergleich zu den häufiger anzutreffenden MIT-lizenzierten Projekten ist Elsa aufgrund der gewählten BSD-3-Variante ein wenig restriktiver. Im Regelfall hat dies jedoch kaum spürbare Auswirkungen. So ließen sich mittlerweile an die 18 unterschiedliche Mitwirkende zu einem Pull Request (PR) hinreißen. Auch Bug Reports oder spezielle Feature Requests sind gerne gesehen. Um einen PR zu erstellen, sollte man sich zunächst mit dem Prozess vertraut machen.

Hier wird auf den recht gebräuchlichen GitHub-Flow gesetzt. Im Unterschied zum Git-Flow zeichnet sich dieser durch sehr häufige Releases und möglichst geringen Overhead aus. Im Fall von Elsa startet die Entwicklung immer im Develop-

Interview mit Sipke Schoorstra, Entwickler von Elsa Workflows

„So ziemlich jedes Backend-System könnte von so etwas profitieren“

Was ist dein Hintergrund?

Sipke Schoorstra: Ich erinnere mich immer noch lebhaft an meinen ersten Computer, den ich an Weihnachten bekommen habe: ein Nintendo Entertainment System. Beigefügt waren jede Menge Spiele, unter anderem Mario Bros, Zelda und The Battle of Olympus. Da wusste ich bereits, was ich mal werden will: Game Designer.

Anschließend sparte ich für meinen ersten Rechner, einen Pentium 100 mit Klik & Play [2]. Ich habe daraufhin Macromedia Director gelernt und mich total in deren Lingo-Skriptsprache vertieft.

Nach einiger Zeit bekam ich ein Jobangebot von einer Firma in Amsterdam. Ich habe deren Angebot angenommen und mein Studium abgebrochen. Das war mein erster richtiger Job als Entwickler.

Ein paar Jahre danach habe ich zusammen mit einem Freund, der meine Leidenschaft für Videospiele teilte, meine eigene Softwareberatungsfirma aufgemacht. Unser Ziel war es, genügend Geld zu verdienen, um eine Spieleentwicklungsfirma zu gründen. So weit ist es noch nicht gekommen, sodass ich derzeit meine Dienste noch als freiberuflicher Softwareentwickler anbiete.

Aktuell besteht meine Aufgabe darin, für eine bekannte Medienfirma in den Niederlanden an einer neuen Microservice-basierten Plattform mithilfe von ASP.NET Core, Kubernetes, Azure und einer Vielzahl anderer Tools und Frameworks zu arbeiten. Das Projekt heißt Videoland und stellt einen On-demand-Videostreaming-Service dar. Mein Team fokussiert sich auf die Onboard User Journey inklusive Sign-up, Abonnements und Rechnungsverwaltung.

Daneben habe ich mit zwei Partnern noch ein kleines Start-up namens Scubaya BV gegründet. Wir arbeiten an neuen Onlinediensten, die hoffentlich im Juli dieses Jahres live gehen.

Warum hast du Elsa erstellt?

Schoorstra: Es gibt eine Vielzahl von Gründen, warum ich Elsa Workflows gestartet habe. Eines meiner vorherigen Projekte hat genauso so eine Workflow-Engine mit einem visuellen Designer für bestimmte Poweruser benötigt. Aufgrund von Einschränkungen bei der Auswahl von Lizenzen konnten wir kein existierendes Projekt (beispielsweise Workflow Engine [3]) verwenden. Unglücklicherweise konnten wir auch nicht die Zeit aufbringen, unsere eigene Workflow-Engine zu schreiben. Am Ende entschieden wir uns, die Windows Workflow Foundation für den MVP zu verwenden, weshalb Benutzer für den visuellen Designer eine kleine Desktop-Applikation installieren mussten. Obwohl das so weit funktionierte, musste ich immer daran denken, wie großartig ein webbasierter Designer für dieses Projekt gewesen wäre.

Doch nicht nur für dieses Projekt schien mir eine Workflow-Engine ziemlich sinnvoll. So ziemlich jedes Backend-System könnte von so etwas profitieren, um Benutzer beispielsweise Standardaufgaben automatisieren zu lassen. Auch ereignisbasierte Aufgaben können damit ohne Quellcodezugriff angelegt und geändert werden. Das modulare Open-Source-CMS-Framework Orchard Core [4] ist hierfür ein tolles Beispiel. Es wird mit einem Workflow-Modul veröffentlicht, das CMS-Nutzern die Möglichkeit gibt, Standardaufgaben von einfachen Formularanfragen bis hin zu komplexen Benutzerprozessen zu automatisieren.

Branch. Dieser sollte auch als Ziel für mögliche PRs verwendet werden.

Wie die meisten Open-Source-Projekte setzt auch Elsa Workflow eine Zusammenarbeit primär über die in den GitHub-Issues geführten Diskussionen voraus. Zwar können PRs ohne vorangehende Diskussion durchaus akzeptiert werden, vor allem bei größeren Änderungen sinkt so aber die schnelle Erfolgsaussicht rapide.

Einschätzung

Mit Elsa gibt es eine wahrlich grandiose Möglichkeit, die Fähigkeiten der Workflow Foundation in .NET-Core-Applikationen wiederaufleben zu lassen. Dank des webbasierten visuellen Designers und der verschiedenen Bereitstellungsmöglichkeiten können Anwendungen sehr einfach von Endanwendern erweitert werden.

Populäre Projekte wie Orchard profitieren bereits von der Integration einer Workflow-Engine, wie auch Elsa-Autor Sipke Schoorstra im Interview erläutert (siehe Kasten). Sein Ziel, mit dem Framework jedem Entwickler solche Möglichkeiten zur Verfügung zu stellen, scheint meiner Meinung nach nicht nur nah, sondern bereits weitestgehend erfüllt zu sein. ■

[1] Elsa Workflows, www.dotnetpro.de/SL2102HiddenGems1
[2] Klik & Play, www.dotnetpro.de/SL2102HiddenGems2
[3] Workflow Engine, https://workflowengine.io
[4] Orchard Core, www.orchardcore.net

Dr. Florian Rappl
ist Solution Architect bei smapiot mit Einsatzgebiet digitale Transformation. Als Microsoft MVP für Entwicklungstools befasst er sich mit Themen wie C#/.NET, TypeScript sowie skalierbaren Backends und Frontends in der Cloud.
@FlorianRappl

dnpCode A2102HiddenGems

Was ich erreichen wollte, war die Wiederverwendbarkeit dieses Moduls in anderen Applikationen, ohne irgendeine Abhängigkeit zu Orchard. Eines Tages begann ich also mit der Arbeit an Elsa. Ohne primären Einsatzbereich wollte ich nur meinen Wunsch erfüllen, einen Satz von Klassenbibliotheken für Workflows zu erstellen, den Entwickler einfach so in ihre eigenen Projekte einbinden können. Das hätte mir selbst in der Vergangenheit viel Arbeit erspart, und bereits da wusste ich, dass es mir viel Arbeit in der Zukunft ersparen würde.

Für welche anderen (Open-Source-)Projekte leistest du noch Beiträge?
Schoorstra: Ich schreibe seit circa 2011 regelmäßig Code für das Orchard-Projekt. Zu dieser Zeit hatte ich bereits einige Kunden, für die ich einige eigenständige CMS auf Basis von ASP.NET Web-Forms erstellt und gewartet habe. Allerdings ist vor allem die Wartung von eigenen Systemen ziemlich aufwendig, weshalb ich nach einem guten Open-Source-CMS gesucht habe. Nachdem ich bereits im Jahr davor viel mit ASP.NET MVC gearbeitet hatte, sollte das neue CMS keinesfalls mit WebForms erstellt worden sein. Ich erinnere mich noch gut daran, wie ich zufällig auf Orchard gestoßen bin.

Ursprünglich von einer kleiner Gruppe innerhalb des ASP.NET-Teams gestartet, wurde Orchard schnell an die Open-Source-Community übergeben, verwaltet durch die Outercurve Foundation. Die Lernkurve war aufgrund der spärlichen Dokumentation ziemlich brutal. Kurz nach dem Start war ich schon fast am Aufgeben und wollte Umbraco als Ersatz wählen. Nachdem ich mich jedoch bereits in Orchards Architektur und Eleganz verliebt hatte, gab ich dem Projekt noch eine Chance. Glücklicherweise fand ich eine Lösung zu meinem Problem. Anschließend fing ich an, über meinen Lernprozess zu bloggen.

Als Ergebnis meiner Open-Source-Tätigkeit wurde ich von Onestop Internet angestellt. Die Anstellung als Orchard-Berater erwies sich insgesamt als eine der besten Entscheidungen meines Lebens. Hierdurch konnte ich noch mehr über Orchard lernen und signifikante Verbesserungen wie beispielsweise das Layouts- und das Dynamics-Forms-Modul beisteuern.

Was ist deine Vision für die Zukunft von Elsa?
Schoorstra: Meine Vision für Elsa Workflows ist, eine tolle Sammlung von Open-Source-Bausteinen zu sein, um Entwickler darin zu bestärken, ihre eigenen Applikationen sehr einfach mit Workflow-Fähigkeiten auszustatten. Diese Workflows sollen sowohl mit Code als auch grafisch über einen webbasierten Designer gestaltet werden können.

Welche Features fehlen aus deiner Sicht noch in C# und .NET?
Schoorstra: Ein Feature, das ich in .NET Core vermisse, ist eine zeitgemäße Workflows-Bibliothek. So etwas wie die Windows Workflow Foundation (WF) im .NET Framework. Wäre ich hier Entscheidungsträger, würde ich den Bau eines webbasierten Workflow-Designers oder von einigen C#-Sprachfeatures zur Erleichterung von Workflows in Code forcieren. Es gibt zwar Ansätze, um WF nach .NET Core zu portieren, allerdings scheint das bislang nicht ganz zu gelingen.