Das Stranglerpattern

Noch viele der heute im Einsatz stehenden Systeme sind schwerfällige Monolithen, die u.a. schlecht skalierbar und wartbar sind. Solche Anwendungen sind in der Regel Kandidaten für ein Refactoring. Oftmals müssen die Systeme während der ganzen Zeit am Laufen sein und trotzdem sollen einzelne Teile kontinuierlich ersetzt werden. Diese Arbeit untersucht anhand eines Content Management Systems (CMS), wie eine Ablösung mit dem Stranglerpattern durchgeführt werden könnte.

Einleitung

Das Stranglerpattern wurde das erste Mal im Jahre 2004 von Martin Fowler in einem Blogeintrag beschrieben [1]. Er schreibt dabei von einem Ausflug in Australien, wo er zusammen mit Cindy die Würgefeigen (engl. Strangler Fig) bestaunte und so zu der Metapher gelangte[1]. Diese Bäume können auf Wirtsbäumen wachsen. Dabei bilden sie sogenannte Luftwurzeln. Erreichen die den Boden, wachsen diese parasitären Bäume noch schneller. Schlussendlich sind rings um die Wirtsbäume mit diesen Wurzeln abgedeckt. Der Wirtsbaum wird sinngemäss erwürgt, weshalb die Würgefeige zu ihrem Namen kam.

Das Pattern

Ähnlich den Würgefeigenbäumen werden mit dem Stranglerpattern einzelne autonome Teilanwendungen gebaut, die kleine Bereiche des Gesamtsystems übernehmen. Während Fowler im Jahr 2004 noch davon schrieb, dass schlussendlich ein neues System ein altes übernimmt, werden die abzuspaltenden Teile heutzutage gerne als Microservices umgesetzt. Da normalerweise nicht alle Teilbereiche einer älteren Anwendung in solche Komponenten abgespaltet werden können, müssen die betroffenen Teile in eine neue Kernapplikation transferiert werden. Somit stellt dieses Pattern vor allem ein mögliches Vorgehen dar, wenn das Programmierte kontinuierlich in das Produktivsystem eingebaut werden muss, da ein Big Bang nicht akzeptabel ist. Ein weiteres Anwendungsgebiet ist, wenn die Kosten um zwei Systeme parallel zu betreiben geringer sind, als wenn nicht so früh, wie möglich mit der Migration auf die neue Plattform begonnen wird.

Als Risiko bei der Anwendung dieses Patterns ist die Gefahr zu nennen, dass das Aufsplitten in die Einführung vieler verschiedener Architekturen führen kann. Es sollte auch darauf geachtet werden, dass die Entwicklung nicht abreisst. Ansonsten ist die Möglichkeit äusserst gross, dass das bisherige archiviert und das alte System repatriiert wird[2].

Das Pattern an einem CMS angewandt

Concrete5 (c5) ist ein herkömmliches CMS, baut auf dem LAMP-Stack auf und wurde als klassischer Monolith programmiert. Mit der am 13.9.2014 veröffentlichten 5.7.0er Version [2] wurde erstmalig das Namespacing in PHP unterstützt, was zu einem kompletten Umschreiben der Codebasis zur Folge hatte. Mit dem nächsten

[1] Wie Fowler im Jahr 2019 revidierte (s. [1], Kapitel Revisions), hat er den Namen von *Strangler Application* zu *Strangler Fig Application* umbenannt. Dies, weil viele Seitenbesuche aufgrund der gewaltätigen Beschreibung, das dem Wort Strangler (Würger) innewohnt, stattfanden.
[2] Für weitere Einzelheiten sei an dieser Stelle auf [5] verwiesen.

Major-Release wurde, neben vielen weiteren Änderungen, die Versionierung auf Semver umgestellt [3], womit die aus dem CMS-Namen stammende 5 aus der Version verbannt wurde. Der erste derartige Release[3] erfolgte am 8.12.2016 mit der Version 8.0.1 [4].

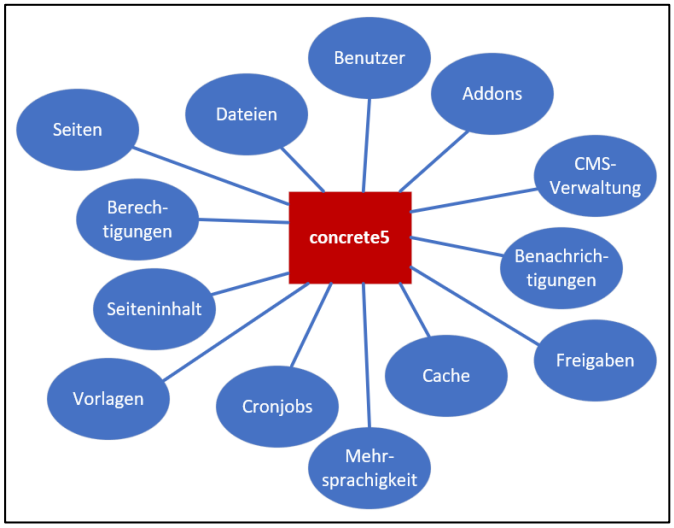

Abbildung 1: Die wichtigsten Elemente eines CMS wie concrete5 (die Anordnung der Elemente ist willkürlich). Nicht aufgelistet sind das Backend und Frontend, wekche nicht als eigenständige Elemente betrachtet werden können, sondern in die einzelnen Elemente eingebetet sind.

Von den wichtigsten Komponenten dieses Systems, wie sie in **Abbildung 1** zu sehen sind, haben sich über die drei Major Releases weder Komponenten dazugesellt noch welche verabschiedet. Mit der Version 8.5.2 jedoch wurde eine REST API Schnittstelle implementiert [5]. An der Architektur des Monolithen hat sich aber nichts verändert.

Eines der Probleme eines solchen Monolithen ist die enge Kopplung der einzelnen Komponenten. Dies führt zu einer verschlechterten Wartbarkeit, was u.a. in nicht tolerierbar langen Release-Zyklen mündet. Ein weiteres Problem stellt die Skalierbarkeit dar, wo es sich nur vertikal skalieren lässt, während eine Horizontale unmöglich ist.

Um nun gemäss dem Stranglerpattern den Monolithen langsam absterben zu lassen, muss ein guter Angriffspunkt gefunden werden, der in einen Microservice verschoben werden kann. Dies sollte wenn möglich eine einigermassen lose gekoppelte Komponente sein. Als solches bietet sich das Datei-Element an. In einem ersten Schritt werden die entsprechenden Dateiklassen durch eine Schnittstelle ersetzt, die die REST API Abfragen verhüllt. Dadurch müssen die Stellen, an denen die bisherigen Dateiklassen verwendet wurden, noch nicht angepasst werden. Gleichzeitig werden die Aufrufe aus dem Frontend, um Dateien hochzuladen oder Metadaten zu ändern auf die neue REST API umgeschrieben. Der über die REST API zugängliche Microservice muss nun das komplette Dateihandling autonom ausführen können. Dabei müssen u.a. die Dateien real auf Festplatten sowie Dateiinformationen in Datenbanken gespeichert werden. Es ist zu prüfen, ob ein Umstieg von Relationalen Datenbanksystemen (RDBMS) zu NoSQL Datenbanken für Komponenten, wie die Dateien, eine akzeptable Alternative darstellt.

Bewährt sich das umgesetzte Konzept nach allfälligen Justierungen im produktiven Einsatz, werden weitere Komponenten nach dem gleichen Muster ersetzt. Als weitere Kandidaten für einen frühzeitigen Umbau sind die Benutzer oder die Berechtigungen. Mit der Erweiterung des Refactorings[4] auf weitere Komponenten muss auch der Umbau der Kernapplikation in Angriff genommen werden. Hier müssen die Elemente eingebaut werden, die nicht sinnvoll ausgelagert werden können. Statt mit den REST API Schnittstellen wird die Kernapplikation jedoch direkt mit den einzelnen Microservices kommunizieren.

[3] Der erste Release der neuen Major-Serie erfolgte fälschlicherweise nur als zweistellige Version (8.0) [5].
[4] Gemäss Birchall sollte hier eher von einem Re-Architekting gesprochen werden [8].

Ein Ablösen des Monolithen darf nicht vor der vollständigen Migration der letzten Komponente durchgeführt werden. Ziel sollte aber sein, dass das Zeitintervall zwischen der letzten Migration und der Ablösung nicht zu gross sein wird.

Diskussion

Durch das Refactoring des Monolithen in kleine Einheiten können die Release-Zyklen deutlich erhöht werden. Es werden nicht alle Komponenten auf RDBMS verzichten können. Somit ist ein zusätzlicher Umgang mit NewSQL-Systemen, um eine horizontale Skalierung zu ermöglichen, unverzichtbar.

Referenzen

[1] M. Fowler, «StranglerFigApplication,» 29 6 2004. [Online]. Available: https://martinfowler.com/bliki/StranglerFigApplication.html. [Zugriff am 6 05 2021].

[2] A. Embler und et al., «Release 5.7.0 · concrete5/concrete5,» 13 09 2014. [Online]. Available: https://github.com/concrete5/concrete5/releases/tag/5.7.0. [Zugriff am 07 05 2021].

[3] T. Preston-Werner, «Semantic Versioning,» [Online]. Available: https://semver.org/. [Zugriff am 06 05 2021].

[4] A. Embler und et al., «Release 8.0.1 · concrete5/concrete5,» 08 12 2016. [Online]. Available: https://github.com/concrete5/concrete5/releases/tag/8.0.1. [Zugriff am 06 05 2021].

[5] G. Starke, A. Heusingfeld, P. Hruschka und et al., «Method Reference,» 03 12 2020. [Online]. Available: http://aim42.github.io/#strangler-approach. [Zugriff am 06 05 2021].

[6] A. Embler und et al., «Release 8.0 · concrete5/concrete5,» 02 12 2016. [Online]. Available: https://github.com/concrete5/concrete5/releases/tag/8.0. [Zugriff am 06 05 2021].

[7] A. Embler und et al., «Connecting to the concrete5 REST API,» 17 5 2019. [Online]. Available: https://documentation.concrete5.org/developers/rest-api/connecting-to-the-concrete5-rest-api. [Zugriff am 06 05 2021].