Konzeptentwicklung einer Serviceorientierten Datenintegration für verteilte Datenbestände

Assignment zum Modul:

Geschäftsprozesse und Anwendungssysteme (ANS40)

15.08.2020

Vladimir Zhelezarov

.....

Studiengang: Digital Engineering und angewandte Informatik - Bachelor of Engineering

AKAD University

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Großunternehmen ABC AG[1] arbeitet in mehreren deutschen Großstädten, wobei der rechtliche Hauptsitz in Stuttgart ist. Die einzelnen Standorte sind aufbauorganisatorisch identisch und produzieren die gleichen Dienste und Güter mit wenigen ortsbedingten Unterschiede. Das Unternehmen ist zu dieser Form gewachsen, durch Zukauf von anderen Unternehmen und der Eröffnung von neuen Niederlassungen. Das hat dazu geführt, dass die einzelnen Standorte sehr selbstständig, arbeits- und funktionsfähig sind, allerdings ist die firmeninterne Kooperation praktisch nicht vorhanden, weil die Kommunikation zwischen den Teilnehmern keine EDV-Unterstützung hat und dadurch langsam und fehleranfällig ist. Eine Aufteilung von Aufträge ist aus dem selben Grund auch nicht möglich. Zusätzlich ist dem Geschäftsführer der Überblick sehr begrenzt, weil Berichte und Unternehmenszahlen nur manuell in verschiedenen Formen gesammelt und analysiert werden.

Eine komplette Umstellung auf Standard- oder Individualsoftware unternehmensweit ist aus finanziellen und organisatorischen Gründen sehr schwer umsetzbar und dadurch auszuschließen. Die vorhandene Software ist Investitionsgut und ist nicht einfach ausschaltbar. Die Umstellung wird auch mit psychologischen Widerständen bei den Benutzern rechnen, wenn das neue System von der alten Software viel zu unterschiedlich und schwer erlernbar ist. Das Ziel dieser Arbeit ist die Erstellung eines Konzepts, das diese zum Teil widersprüchlichen Konditionen erfüllt.

Als erstes werden die genauen Anforderungen präzisiert. Es wird klar, dass die Software die Geschäftsprozesse abbilden muss, um sie besser zu unterstützen. Wie dadurch eine bessere Auslastung der Ressourcen erzielt wird, zeigt der Paragraph zur prozessorientierten Geschäftskonstellation. Diese Funktion wird erreicht durch eine Aufteilung von Verantwortungen auf Services, die miteinander und mit einem zentralen Server kommunizieren. Der Zentralserver hat auch zusätzliche Funktionen, die die Geschäftsprozesse unterstützen. Nach dieser Übersicht muss das System etwas näher betrachtet werden, mit den Themen Datenspeicherung und Kommunikation. Die Mitarbeiter wollen auch ein geeignetes Mittel zur Verfügung, um miteinander zu kommunizieren oder Daten auszutauschen. Ein möglicher Ansatz dafür betrachtet der Paragraph zur Mitarbeiterkommunikation.

Auch wenn aufgrund des begrenzten Umfangs die Klärung mancher Einzelheiten in dieser Arbeit keinen Platz findet, betrachten wir kurz manche von besonderer Bedeutung wie der Aufbau und die Funktion der Schnittstellen, die Versionierung als Instrument für Abwärtskompatibilität und wie wir mit den potenziellen Ausfällen umgehen wollen.

Das ganze muss auch umgesetzt werden. Wie die Planung aussehen kann diskutiert der letzte Paragraph vom Hauptteil.

[1] Fantasiename

2 Grundlagen

2.1 Begriffe

Grundprinzip des vorgestellten Konzepts ist die Service-Orientierte Architektur. Weil es dafür verschiedene Definitionen bei den verschiedenen Autoren gibt, verstehen wir in dieser Arbeit diese Architektur im allgemeinen Sinn als die Zusammenwirkung von lose gekoppelten Diensten (Services). Jeder Dienst erfüllt seine Aufgaben wie er es am besten findet, indem er sich um Implementierung und Einzelheiten selber kümmert. Die Kommunikation mit der Außenwelt erfolgt nur über klar definierten Schnittstellen[1].

In unserer Architektur werden keine Schnittstellen-Definitionen ausgetauscht, die vorhandenen Services sind schon vordefiniert und es gibt praktisch keine gemeinsam genutzte Ressourcen. Somit kann sie als eine Mikroservice-Architektur bezeichnet werden[2].

Als Software-Kopplung bezeichnen wir die Abhängigkeit zwischen den beiden Kommunikationsteilnehmern - je stärker die Kopplung, desto wahrscheinlicher dass eine Änderung bei einem Teilnehmer zu einer Fehlfunktion bei dem anderen führen wird. Lose Kopplung ist ein sehr wichtiges Merkmal zu erzielen[3];

Unter „Knoten“ ist die Software einer Abteilung auf einem konkreten Standort zu verstehen, die nach außen wie eine Einheit aussieht und funktioniert. Ein Knoten kann auch mehrere Dienste anbieten;

„System“ oder „Software-System“ bezeichnet die Software, die auf den einzelnen Knoten und auf dem Zentralserver verteilt ist, die die Funktion des vorgestellten Konzept anbietet.

2.2 Anforderungen

- Die Umstellung darf nicht eine Anbindung an einem bestimmten Anbieter als Folge haben (vendor lock); Zusätzlich muss die Software ohne große Hürden auch nach dem Ausstieg der originalen Entwickler weiter gepflegt werden;
- Die normalen Arbeitsabläufe des Unternehmens dürfen während der Umsetzung nicht unterbrochen werden;
- Es muss die Möglichkeit bestehen, dass alte oder noch nicht geänderte Systeme weiter arbeiten können, auch wenn sie die neu angebotenen Funktionen nicht nutzen;
- Die einzelnen Abteilungen müssen weiterhin unabhängig voneinander arbeiten können;

[1] Vgl. Bass et al. (2015), S.222
[2] Vgl. Richards (2016), S.43
[3] Newman (2015), S.30

- Eine Strategie, die mit dem Ausfall von Kommunikationsteilnehmer umgeht, muss vorhanden sein.

Zusätzlich gelten natürlich auch die allgemeinen Kriterien für eine qualitative Software: Funktionalität, Zuverlässigkeit, Effizienz, Benutzbarkeit, Übertragbarkeit, Änderbarkeit, Testbarkeit und Transparenz[1].

3 Konzepterarbeitung

3.1 Zur prozessorientierten Unternehmenskonstellation

Das Unternehmen besteht aus eigenständigen Standorten, die weitgehend unabhängig voneinander funktionieren. Jeder hat eine eigene Produktion, ein eigenes Rechnungswesen usw. Der Austausch zwischen den Standorten ist aufgrund mangelnder EDV-Unterstützung praktisch nicht vorhanden.

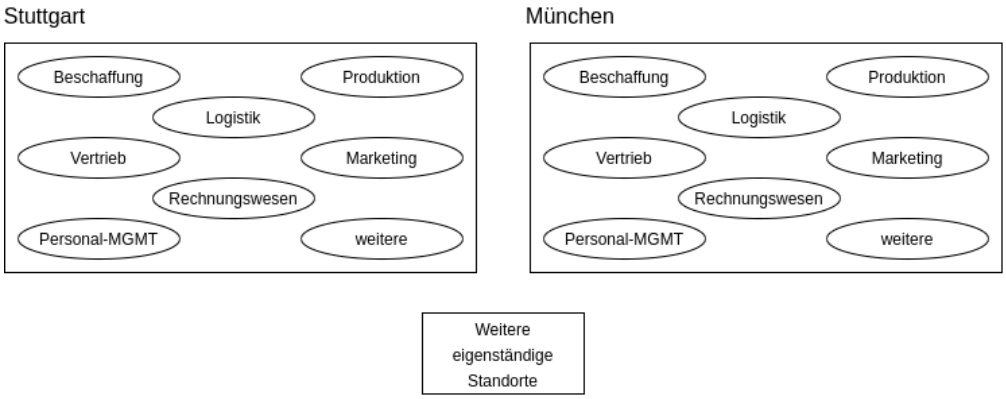

Abbildung 1: Standortorientierte Geschäftskonstellation

Diese Eigenständigkeit bringt mit sich die Freiheit die eigenen Prozesse am Standort selbstständig zu gestalten, dafür gibt es jedoch ein paar Probleme, die sich zusätzlich bei dem wachsendem Unternehmen noch stärker auswirken:

- Der Überblick für die Konzernleitung ist nur begrenzt vorhanden. Viele Daten müssen manuell angefragt, angenommen und analysiert werden. Das erschwert natürlich die Entscheidungsträger bei der Konzernpolitik;
- Auch wenn identische Güter und Dienstleistungen angeboten werden, ist die interne Kooperation schwer bis gar nicht umsetzbar. Im Extremfall würde das dazu führen, dass bei einem Standort Aufträge abgesagt werden oder Termine nicht eingehalten werden, während bei dem anderem Standort Kurzarbeit angesagt wird;

[1] Hoffmann (2013), S.7

- Auch im Rahmen des eigenen Standorts kann es zu Engpässen bei manchen Abteilungen kommen, wobei andere kaum etwas zu tun haben.

Mithilfe von gut ausgelegten EDV-Systemen kann das Unternehmen mit mehr Mitwirkung und Verteilung der Aufgaben innerhalb des Prozessenumfelds arbeiten[1]. Aus der Sicht des Auftraggebers, auch wenn dieser extern (Kunde) oder intern (andere Abteilung) ist, gibt es nur Prozesse.

Abbildung 2: Prozessorientierte Geschäftskonstellation

In dieser Form kann leicht erkannt werden, dass die oben genannten Probleme gut umgangen werden. Allerdings ist dieses Konzept nicht ohne Anpassungen anwendbar. Wenn es nur um die Bearbeitung von Informationen geht - typische Beispiele sind Rechnungswesen, großenteils Marketing, Statistik usw. - ist die Konstellation direkt anwendbar und bringt auch den besten Nutzen. Für den Fall, wenn Güter erstellt und bewegt werden, oder wenn es um Personen geht müssen die damit verbundene Zeit und Geldkosten betrachtet werden. Im System kann dies durch Schrittrechnungen umgesetzt werden, indem immer weitere Standorte abgefragt werden und die zusätzlichen Kosten miteinkalkuliert werden.

Das System unterscheidet dabei zwei Fehlkonditionen:

- Nirgendwo sind genug Kapazitäten vorhanden. Menscheneingriff ist notwendig - eventuell ist der Auftrag neu zu verhandeln und im schlimmsten Fall abzusagen;
- Die Herstellungskosten inklusive die extra Zeit- und Geldkosten überschreiten die geplanten Grenzen. Menscheneingriff ist wieder wie im vorherigen Fall notwendig.

Es gelten folgende Besonderheiten:

- Der Auftrag muss auf Aufgaben zerlegbar sein. Wenn das nicht der Fall ist, wird der ganze Auftrag als eine Einheit gesehen und bearbeitet;

[1] Vgl. Newman (2015), S.31

- Die zusätzlichen Zeit- und Geld-Kosten müssen voreingestellt oder kalkulierbar sein. Beim Anlegen des Auftrags werden dem Mitarbeiter aus dem Vertrieb Standartwerte angeboten, die er an seiner Entscheidung anpassen kann;

- Vordefinierte Grenzen an Kosten und Dauer. Diese werden vom Vertrieb beim Anlegen des Auftrags vorgesehen.

Eine Darstellung des ganzen Ablaufs einer Auftragsbearbeitung befindet sich in den Dateien *Auftragsbearbeitung* im Anhang zu dieser Arbeit.

3.2 Dienste / Services

Es ändert sich nichts wesentlich für die Mitarbeiter, was zusätzliche Hilfe leisten kann bei der Umstellung zum neuen System: Finanzbuchhaltung bleibt Finanzbuchhaltung. Was neu gekommen ist, ist dass manchmal Aufgaben von den anderen Standorten angenommen werden.

Es stellt sich die Frage ob die Dienste noch feiner zerteilt werden sollten, allerdings würde das dazu führen, dass auch die Geschäftsprozesse angepasst werden müssen. Auch, bei einer zu feinen Zerteilung wird den Mitarbeitern eine kleinere Auswahl von Aufgaben angeboten, wodurch die Motivation leidet. Eine zu feiner Zerteilung würde allgemein zu mehr Komplexität und Aufwand führen[1].

Die Aufgaben die ein Service ausführt, teilen wir in zwei Typen auf:

- vordefinierte, klar berechenbare Aufgaben - diese werden ohne Menscheneinfluss direkt vom System verwaltet und vergeben. Die ausgewählten Mitarbeiter bekommen eine Nachricht mit den zugeteilten Aufgaben und Links zu den benötigten Ressourcen;

- spezifische Aufgaben, die Menscheneinfluss benötigen, um zugeteilt zu werden. Der Abteilungsleiter bekommt eine Anfrage auf Ressourcen. Er kann entscheiden ob überhaupt seine freien Personal-Ressourcen dafür reichen, wer und womit beauftragt wird.

Das System pflegt Listen für jeden Standort, in denen steht wer und womit beschäftigt ist.

[1] Vgl. Daya et al. (2015), S.32-33

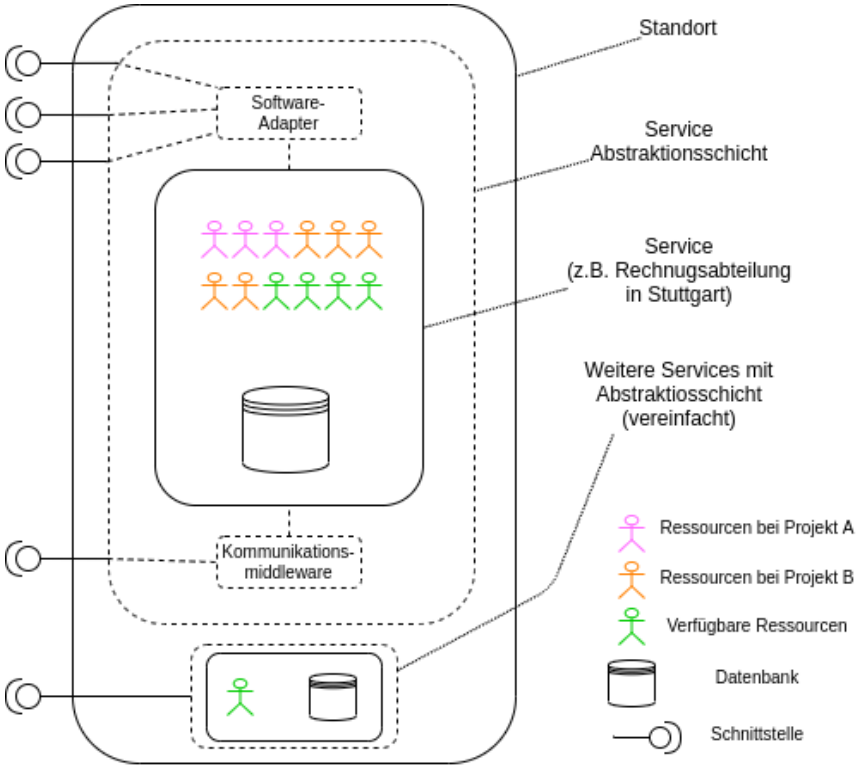

Abbildung 3: Konzept der Dienste

3.3 Zentralserver

Das vorgestellte Konzept sieht ein Zentralserver in Stuttgart vor. Dieser hat folgende Funktionen:

- koordiniert die Aufträge, deren Verteilung und Fortschritt;
- pflegt eine vollständige Kopie vom gesamten Datenbestand des Unternehmens. Diese wird täglich aktualisiert;
- koordiniert die Datensuche und hilft dabei mit eigenem Cache;
- führt allgemeine Suchen im gesamten Datenbestand durch. Das ist für Dienste vorgesehen, die nicht unbedingt die aktuellsten Daten haben wollen - z.B. Statistik;
- überwacht alle Knoten und alarmiert den Netzwerkadministrator, wenn einer offline ist.

Diese Funktionen werden in den folgenden Punkten auch weiter betrachtet.

3.4 Datenspeicherung

Es werden keine Änderungen bei den vorhandenen Datenbanken vorgesehen. Als Extra-Schicht wird sich die neue Software um die Kommunikation zwischen den verschiedenen Standorten und dem Zugang auf die nicht-eigenen Datenbestände kümmern. Die Knoten bilden somit eine Art von Ressourcen-basierte Architektur[1].

Weil die Netzwerkanfragen zeitintensiv sind, sind Caches vorgesehen - sowie bei den Clients, als auch bei dem zentralen Server. In einem Cache werden Datensätze in der Form von Primärschlüssel nach Datenbanken - Adresse der Datenquelle für die letzten X Zugriffe gespeichert. X kann z.B. mit 1000 anfangen und kann dann je nach Netzverkehr und Hardware-Verfügbarkeiten geändert werden.

Wenn ein Datensatz benötigt ist, wird zuerst die lokale Datenbank abgefragt, dann der lokale Cache, erst dann wird eine Netzanfrage am zentralen Server geschickt. Dieser folgt ähnlichen Ablauf indem er zuerst in seinem lokalen Cache sucht, dann in der lokalen Datenbankkopie und als letzte Lösung werden alle Standorte abgefragt.

Wenn ein neuer Datensatz erstellt werden muss, muss zuerst geprüft werden, ob dieser schon irgendwo existiert. Wenn nicht, dann wird dieser in der lokalen Datenbank erstellt.

Bei dem zentralen Server wird neben dem lokalen Cache auch eine vollständige Kopie aller Datenbanken vorgesehen. Einmal täglich, außer der Geschäftszeiten, wird diese gemeinsame Datenbank mit allen einzelnen Standorten synchronisiert. Sie wird benutzt als Backup und für allgemeine Datenbankanfragen. Manchmal kann es notwendig sein nach anderen Kriterien zu suchen als dem Primärschlüssel. Für diesen Fall wird die Datenbankkopie auf dem zentralen Server abgefragt. Dieses Vorgehen entlastet die Netzwerkkommunikation und minimiert den Aufwand für die Software, hat jedoch den Nachteil, dass die Datenbestände nicht die aktuellsten sein können.

Interessant ist bei verteilten Datenbeständen wie im hier betrachteten System die Möglichkeit, dass die Kommunikation wegen Netzproblemen oder aus einem anderem Grund ausfällt. Bei über dem Netzwerk verbundene Software darf diese Möglichkeit nicht ausgeschlossen werden[2]. Um den Einfluss auf die Geschäftsprozesse zu minimieren, kann die Lösung wie folgt aussehen:

- Im Fall einer neuen Datenerstellung erstellt das System die Daten trotzdem in der lokalen Datenbank. Dieser Datensatz wird in der Software gemerkt. Gleichzeitig wird eine Fehlermeldung ausgelöst. Wenn die Kommunikation wieder aufgebaut wird, vergleicht der zentrale Server alle markierte Datensätzen, die sich auf die selbe Daten beziehen, und übernimmt den ältesten davon. Alle anderen werden benachrichtigt den Datensatz zu löschen. Dieses Vorgehen kann natürlich zu Datenverlust führen. Beim Benutzer

[1] Vgl. Van Steen; Tanenbaum (2018), S.64
[2] Rotem-Gal-Oz (2006), Internetquelle

wird eine Info angezeigt, in der er prüfen kann, ob im übernommenen Datensatz etwas fehlt;

- In allen anderen Fällen bleibt das System „hängen“ indem es immer wieder versucht die Kommunikation wieder herzustellen. Beim Benutzer wird eine Fehlermeldung angezeigt sodass er sich mit etwas anderes beschäftigen kann, bis der Fehler behoben wird. Die typische Situation für diesen Fehler ist wenn ein Mitarbeiter mit Daten aus einem anderem Standort arbeitet. Weil bei jedem Auftrag die lokalen Personalressourcen bevorzugt werden, kann der externe Mitarbeiter ohne viel zu großen Folgen für gewisse Zeit für das Projekt nicht verfügbar sein.

Eine Darstellung dieser Konzepte befindet sich in den Dateien *Datensuche_Knoten* und *Datensuche_Server* im Anhang zu dieser Arbeit.

3.5 Software-Kommunikation

Die zahlreichen Anwendungen, die im Unternehmen benutzt werden, benutzen verschiedene, manchmal proprietäre Datentypen oder auch Begriffe mit nicht herkömmlicher Bedeutung. Sogar Anwendungen, die den gleichen Standards folgen, können inkompatibel zueinander sein[1]. Als Software-Adapter benutzt das Konzept Plug-Ins, Erweiterungen oder eigenständige Anwendungen. Diese wandeln die Ausgabe von einer Software so um, dass sie nur die vereinbarte Begriffe in entsprechender Form benutzt.

Ein direkter Zugriff auf fremde Datenbestände ist ausgeschlossen um die Koppelung zu minimieren. Alle Zugriffe erfolgen über die Schnittstellen. Diese werden von den Software-Adapter angeboten. Die Adapter kommunizieren direkt mit der vorhandenen Software auf dem Standort. Hierbei muss auch auf Authentifizierung und Zugriffsrechte geachtet werden.

Damit das System funktioniert ist es nötig, dass alle Knoten dieselbe „Sprache“ sprechen. Darum ist es sehr sinnvoll das ganze Projekt mit einem oder mehreren Terminen anzufangen, an den alle Betroffenen oder Interessierten diskutieren können, was für Daten und Informationsflüsse es im Unternehmen gibt und wie diese ab sofort unternehmensweit genannt werden sollen. Es ist auf keinen Fall notwendig und vielleicht auch kontraproduktiv, wenn die IT-Leute das alleine machen, weil sie nicht unbedingt einen Überblick über alle möglichen Prozesse im Unternehmen verfügen. Um den Aufwand in Grenzen zu halten können sich diese Treffen auf Abteilungen beziehen.

[1] Melzer (2010), S.71

3.6 Schnittstellen

Manche in der Praxis gut etablierten Ansätze bei dem Aufbau einer Schnittstelle sind wie folgt:

- Der Client ruft Funktionen auf, die auf dem Server ausgeführt werden. Client und Server sind stark aneinander gekoppelt; Geschäftslogik ist auf beiden zu finden. Ein typischer Vertreter hier ist der RPC Ansatz. Eine Änderung der einen Seite führt zwangsläufig zu einer Fehlfunktion der anderen[1];

- Vordefinierter API-Vertrag - Client und Server wissen genau was erwartet und was gegeben ist. Die tatsächliche Implementierung ist hinter der Schnittstelle verborgen und somit ist die Koppelung minimiert. Als Vertreter können hier klassische SOA mit WSDL-Dateien[2] und der etwas neuere Ansatz von Google gRPC mit Protobuf[3] genannt werden;

- Die gesamten Daten und Logik sind ausschließlich auf dem Server und die Clients werden möglichst allgemein geschrieben. Das führt zu der stärksten Entkopplung. Als Nachteil ist der erhöhte Datenverkehr, weil die Clients so wenig wie möglich über den Ressourcen wissen dürfen und dadurch mehr Fragen stellen müssen. Dieser Ansatz wird von der REST Architektur[4] gefolgt;

- Der Server wird als Datenquelle betrachtet und die Clients wissen alles über die vorhandenen Daten. Dadurch können sie sehr präzise Anfragen stellen und der Kommunikationsaufwand wird minimiert. GraphQL von Facebook[5] verfolgt diesen Ansatz;

Jeder Ansatz hat Vor- und Nachteile. Für das hier vorgestellte System wird gRPC aus folgenden Gründen gewählt:

- Das benutzte Protokoll Protobuf ist sehr sparsam an Netzwerkbelastung, weil die Daten stark komprimiert in einem binären Format versendet werden[6] - also hervorragend für schlecht ausgebaute Routen wo viel gesendet wird;

- Der Aufbau von Client- und Server-Stubs wird automatisiert nach XML-ähnlichen Definitionsdateien gemacht[7] und entlastet somit die Entwickler;

[1] Vgl. Van Steen; Tanenbaum (2018), 173-174
[2] Vgl. Melzer (2010), S.115
[3] https://grpc.io/docs/what-is-grpc/core-concepts/, (Zugriff am 15.08.2020)
[4] Vgl. Fielding (2000), S.76ff
[5] http://spec.graphql.org/June2018/, (Zugriff am 15.08.2020)
[6] https://developers.google.com/protocol-buffers/docs/encoding, (Zugriff am 15.08.2020)
[7] https://developers.google.com/protocol-buffers/docs/overview#generating, (Zugriff am 15.08.2020)

- Klare API-Verträge in den .proto Dateien[1]. Es ist für jeden Teilnehmer eindeutig klar, was die Schnittstelle anbietet;
- Authentifizierung und Verschlüsselung sind direkt unterstützt[2];
- gRPC benutzt HTTP2[3] mit den damit verbundenen Vorteilen wie Multiplexing, Kompression und schnellere Verbindungen[4].

Die Nachteile sind[5]:

- Das binäre Format ist nicht von Menschen lesbar. Weil unser System eine Bearbeitung von Menschen nicht vorsieht ist das nur ein Problem für eine potenzielle Fehlersuche, in welchem Fall geeignete Tools benutzt werden können;
- Keine direkte Unterstützung von Parsing mit dem Browser - das wird auch vom vorgestellten System nicht vorgesehen.

sowie auch:

- Nicht so lose gekoppelt wie REST. Allerdings ist echte REST mit HATEOAS schwer zu implementieren und zusätzlich sehr belastend für das Netzwerk[6];
- Keine Änderung der Kommunikation ohne Änderung im API-Vertrag. Schließlich ist das ein Vorteil, weil so jeder sich darauf verlassen kann, dass der Vertrag eingehalten wird. Erweiterungen der Kommunikation können durch Versionierung gelöst werden, wie im nächsten Punkt diskutiert wird.

3.7 Umgang mit Änderungen

Es kann angenommen werden, dass die Daten und Aktivitäten sich ändern werden. Folgende Ereignisse können ein Bedarf an Änderungen als Folge haben:

- Der Gesetzgeber fordert neue Informationen, oder ändert die Form in der die geforderten Informationen zur Verfügung gestellt werden müssen;
- Die Firma fängt mit einer ganz neuen Produktlinie an. Wenn die Software flexibel genug ist, ist z.B. ein neues Produkt kein Grund etwas zu ändern, eine ganze neue Produktlinie mit ganz neuen Besonderheiten aber schon;

[1] https://developers.google.com/protocol-buffers/docs/overview#services, (Zugriff am 15.08.2020)
[2] https://grpc.io/docs/guides/auth/, (Zugriff am 15.08.2020)
[3] The Linux Foundation (2020), Internetquelle
[4] Vgl. Belshe et al. (2015), Internetquelle
[5] Vgl. Newton-King (2019), Internetquelle
[6] Vgl. Torikian et al. (2016), Internetquelle

- Die Geschäftsprozesse werden optimiert. Wenn eine Änderung der Geschäftsprozesse zur besseren Wettbewerbsfähigkeit führen wird, muss die Software mitspielen und nicht im Weg stehen.

Wenn also davon ausgegangen werden kann, dass Änderungen kommen werden, ist die nächste Frage wie diese Änderungen bei allen möglichen beteiligten Servern und Clients durchgesetzt werden. Dass bei jeder Änderung das ganze Unternehmen stillgelegt wird, bis die IT die gesamte Software aktualisiert, ist sicherlich eine Katastrophe aus jeder Sicht.

Im Idealfall wären alle Änderungen bei einer neuen API-Version rückwärts-kompatibel. Allerdings kann das leider nicht immer garantiert werden. Darum benutzt das System gleichzeitig mehrere Versionen von den Schnittstellen. Der Client meldet in seiner Anfrage auf welcher Schnittstellen-Version sich seine Anfrage bezieht.

Der ausgewählte Ansatz von gRPC unterstützt Versionierung mit dem *package* Schlüsselwort[1].

3.8 Überwachung

Die Zusatzschichten bei der Software bringen den Vorteil mit sich, dass die darunter liegende Legacy-Software weiter benutzt werden kann. Zusätzlich bieten sie zusammen mit dem Zentralserver neue Funktionen an, die die Geschäftsprozesse unterstützen. So wird alles zu einem verteilten System, das aufgrund von Netz- oder Kommunikationsfehlern teilweise ausfallen kann. Der Zentralserver hat die größte Belastung an Netzverbindungen und kümmert sich zusätzlich um die zentrale Datenbankkopie. Dadurch ist er noch empfindlicher auf solche Störungen.

Wenn alle einzelnen Knoten von Menschen überwacht werden, kann das schnell zu einem unvertretbarem Aufwand führen. Dafür ist eine Überwachung nur am zentralen Server vorgesehen. Der Zentralserver fragt auch regelmäßig alle Knoten ab, damit das Vorhandensein ihrer Funktion und Erreichbarkeit geprüft wird. Wenn ein Knoten in einem bestimmten Zeitraum nicht antwortet, bekommt der Netzwerkadministrator eine Fehlermeldung. Wenn eine Adresse wieder online ist, wird geprüft ob in der Zwischenzeit da neue Datensätze erstellt oder gelöscht wurden. Eine eventuelle Änderung der Daten hat keine Konsequenzen - die andere Teilnehmer wollen nur wissen wo sich ein bestimmter Datensatz befindet und nicht woraus er besteht.

Zusätzlich schicken die Knoten selbstständig Fehlermeldungen, wenn sie mit einem anderen Knoten nicht kommunizieren können.

Eine Darstellung von diesem Ansatz ist in den Dateien *Heartbeat* im Anhang zu dieser Arbeit zu finden.

[1] https://developers.google.com/protocol-buffers/docs/reference/proto2-spec#package, (Zugriff am 15.08.2020)

3.9 Mitarbeiter-Kommunikation

Tools für die zwischenmenschliche Kommunikation sind mit dem neuen System noch wichtiger, weil die kooperierenden Mitarbeiter jetzt nicht mehr unbedingt am selben Standort sind.

Eine mögliche Option wäre auch dafür eine eigene Lösung zu entwickeln mit dem dazu gebundenem Zeit- und Kostenaufwand. Weil die Kommunikation eine sehr typische Problematik darstellt, können hier fertige Lösungen herangezogen werden. Die Anforderungen können so aussehen:

- Die Kommunikation beinhaltet firmeneigene und empfindliche Daten und muss auf jeden Fall gesichert erfolgen. Sogar der Dienstanbieter darf kein Zugang darauf haben;
- Es müssen verschiedene Arten von Nachrichten und Multimedia transportiert werden - Textnachrichten, Audio- und Videodaten, sowie auch allgemeine Dateien;
- Die Software muss leicht bedienbar sein und auf verschiedenen Betriebssystemen und Hardware funktionieren;
- Jede Person in der Firma muss leicht auffindbar sein.

Als Nachteile einer fertigen Software können folgende Punkte gesehen werden:

1. Abhängigkeit von einer externer Firma für Aktualisierungen/Fehlerbehebungen;
2. Vertrauen zur Firma;
3. Keine eigene Anpassungen oder Sonderwünsche möglich.

Punkt 1 kann bei Standardsoftware nicht vermieden werden. Die mögliche Gefahr wenn die Software nicht funktioniert kann durch geeignete Alternativen minimiert werden. Weil es gerade Standardsoftware ist und dazu auch mit sehr vielen Anbieter, kann leicht auf eine andere Standardsoftware umgestiegen werden.

Das Vertrauen aus dem Punkt 2 kann durch Analyse der Marktposition und der Kundenzufriedenheit/Bewertungen vom externen Anbieter aufgebaut werden. Eine interessante Option ist die Nutzung von Open-Source Software. Der Code ist verfügbar zum Analysieren und es ist kein blindes Vertrauen mehr nötig, um zu wissen, ob versteckte Überraschungen im Code sind. Als zweiter Vorteil ist die Möglichkeit für die eigene IT bei der Verbesserung vom Code mitzumachen.

Zum Punkt 3 kann auch ein Vorteil sein, wenn die Lösung eine Open-Source Software ist. Durch eine Verzweigung im Code-Baum (branching) können die eigenen Anpassungen zusammen mit dem Hauptcode gepflegt werden.

Als Bonus-Vorteil einer Open-Source Lösung dürfen auch die Kosten nicht vergessen werden und dem damit verbundenem leichterem Umstieg auf eine andere Lösung.

In dieser Arbeit wird auf eine konkrete Empfehlung verzichtet, weil die Auswahl groß genug ist, und sie letztendlich auch von persönlichem Geschmack abhängt.

3.10 Planung der Umstellung

Eine „Big-Bang“ Umstellung ist sehr fehleranfällig und potenzielle Fehler können schwere Folgen für das Unternehmen haben[1]. Dies ist von der Leitung nicht erwünscht und auch nicht nötig. Das neue System ist als eine Oberschicht auf der alten Software gedacht, die neue Funktionen zur Verfügung stellt. Darum kann das Projekt schrittweise umgesetzt werden.

1. Ausgewählte Abteilungen auf zwei Standorten werden mit Adaptern, Schnittstellen, Kommunikationssoftware versehen. Diese kommunizieren nur direkt miteinander ohne einen Zentralserver und mit begrenzter Funktionalität;

   (a) parallel zu allen Punkten laufen regelmäßig Besprechungen mit den betroffenen Mitarbeitern um Probleme zu finden, oder neue Ideen für Verbesserungen zu bekommen.

2. Aufbau und Inbetriebnahme vom Zentralserver. Als erster Schritt kann nur eine Kopie der Datenbanken von den Standorten in Phase 1 getestet werden. Wenn diese Funktion vorhanden ist, ist die Kommunikation mit den Knoten zu testen;

3. Weitere Knoten/Standorte schrittweise zufügen;

   (a) ab diesem Schritt werden keine Verbesserungen mehr implementiert, nur Problembehebungen.

4. Mitarbeiter-Kommunikationsprogramm bei ausgewählten Mitarbeitern als zweite Software benutzen. Wenn erfolgreich, unternehmensweit umsetzen.

5. Endgültige Einschulung der Mitarbeiter und die Softwareadministratoren.

[1] Vgl. Melzer (2010), S. 57

4 Zusammenfassung

Das Wachstum von ABC AG bringt viel Potenzial mit sich, ist aber auch mit Herausforderungen verbunden. Das Zusammenspiel im Unternehmen wird gleichzeitig immer wichtiger und auch immer schwieriger. Alte Ansätze, die viel auf Menscheneingriffen basieren sind nicht mehr realistisch aufgrund der stark erhöhten Menge und Komplexität der Datenflüsse. Genau solche Aufgaben sind bestens für Software geeignet, weil sie gut mit Informationsflüssen umgehen kann. Der Zwang zur Entwicklung eines solchen Softwaresystem kommt nicht nur aus den internen Schwierigkeiten die Daten zu steuern, sondern auch weil diese Problematik schon die Wettbewerbsfähigkeit gefährdet.

Das in dieser Arbeit vorgestellte Konzept basiert auf eine starke Zusammenwirkung aller Beteiligten, und baut auf Prozessorientierung auf, als der logische Weg, um bessere Kooperation zu erreichen. Somit werden Probleme wie die ungleichmäßige Belastung von Unternehmensressourcen oder die ungenügende Übersicht der im Unternehmen laufenden Prozesse minimiert. Die Verteilung von Aufträgen wird zentral gesteuert, ohne dabei Menscheneingriff komplett auszuschließen, wenn Anpassungen im Auftrag notwendig sind.

Die vorhandene Software wird nicht außer Betrieb genommen, weil das enorme Kosten und Probleme bei der Umstellung mit sich bringen kann. Stattdessen werden alle Altanwendungen mit Adaptern und Middleware versehen, womit diese über klar definierten Schnittstellen kommunizieren können. Die Abteilungen in den verschiedenen Standorten werden zu Diensten, die Aufgaben beantragen oder auch selber ausführen können. Als eine zentrale Stelle, die alle Kommunikationen und Abläufe synchronisiert ist ein Zentralserver in Stuttgart geplannt. Dieser hilft bei der Datensuche unternehmensweit, pflegt eine lokale Kopie der gesamten Datenbestände des Unternehmens und kontrolliert die Erreichbarkeit aller Knoten. Zusätzliche Funktionen wie eine allgemeine Datensuche erleichtern Prozesse wie die Statistik.

Die Daten bleiben auch dort, wo sie am häufigsten benötigt sind, nämlich bei der Abteilung, die sie erstellt hat. Die Suche nach Daten erfolgt nach Wahrscheinlichkeitsprinzip - zuerst lokal und dann immer ein Schritt weiter und allgemeiner. Somit werden Verzögerungen und Kommunikationsaufwand minimiert. Caches, die lokal und beim Zentralserver angeboten sind, helfen bei der Suche. Die Knoten kommunizieren nur über die Schnittstellen. Damit es einen gemeinsamen Grund für den Datenaustausch gibt, sind Termine und Absprachen vor der Inbetriebnahme vorgesehen, indem unternehmensweit eindeutige Begriffe definiert werden. Diese bilden die gemeinsame Sprache, die im Firmennetz gesprochen wird. Die genannten Schnittstellen bilden die Brücke zwischen dem verteilten System und den unterschiedlichsten Anwendungen vor Ort. Sie werden durch Software-Adapter realisiert.

Für ein System dieser Größe ist eine Überwachung nicht wegzudenken. Um Kosten und Aufwand zu minimieren wird diese nur aus Stuttgart zentral gesteuert.

Auch für die interne Kommunikation zwischen Menschen ist gedacht. Mit einer Analyse

der Vor- und Nachteile wurde auf Open-Source Standardsoftware hingewiesen, weil diese leicht einzusetzen oder wechseln ist.

Um die gesamte Umstellung konkret im Unternehmen durchzuführen, sind Phasen geplant, in welche Schritt für Schritt die neue Software eingesetzt wird. Die Einschulung der Mitarbeiter läuft dabei parallel ab und in den ersten Phasen sind neben der Problembehebung auch Verbesserungen durch Ideen von Mitarbeitern geplant.

Um die genannten Optimierungen zu erreichen und gleichzeitig den Aufwand und die Kosten in Grenzen zu halten wurden auch Kompromisse gemacht:

- Die angebotenen Dienste sind alle vorprogrammiert und das System geht davon aus, dass alle anderen Knoten gleich aufgebaut sind. Würde auf einem Standort ein neuer Dienst eingeführt werden, wird es schwierig sein alle anderen darüber zu informieren oder dass sie den neuen Dienst benutzen;

- Es kann zu Spannungen zwischen den Mitarbeitern kommen, wenn ein Auftrag verteilt wird. Es kann sein, dass Mitarbeiter die lokalen Kollegen bevorzugen und die anderen mehr belasten;

- Der zentrale Server hat mehrere wichtige Funktionen, dadurch wird das ganze System auf seinem potenziellen Ausfall sehr empfindlich. Die Überwachung, die auch in Stuttgart vorgesehen ist, kann allerdings dabei helfen.

Weitere Planung und Entwicklung ist notwendig, um die vorgestellten Ergebnisse weiter zu verbessern. Das vorgestellte Konzept bildet ein Gerüst, das auf keinen Fall als endgültig oder komplett betrachtet werden darf, sondern als Wegweiser um die optimalen Kompromisse bei der konkreten Problemstellung zu finden. Die Betonung dieser Arbeit lag auf der gemeinsamen Datenspeicherung und Datenaustausch. Die anderen Themen und Wünsche des Unternehmens wurden nur oberflächlich behandelt. Weitere Forschung überschreitet aber den hier verfügbaren Umfang.

Literaturverzeichnis

Bass, L.; Clements, P.; Kazman, R. (2015)
Software architecture in practice, Upper Saddle River, NJ et al.

Belshe, M.; Peon, R.; Thomson, M. (2015)
Hypertext transfer protocol version 2, https://http2.github.io/http2-spec/ (Zugriff am 15.08.2020)

Daya, S. et al. (2015)
Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach, IBM Redbooks, o.O.

Fielding, R. T. (2000)
REST: architectural styles and the design of network-based software architectures, Dissertation, University of California, Irvine

Google Inc. (o.J.)
Protocol Buffers, https://developers.google.com/protocol-buffers (Zugriff am 15.08.2020)

Hoffmann, D. W. (2013)
Software-Qualität, Berlin, Heidelberg

Melzer, I. (2010)
Service-orientierte Architekturen mit Web Services: Konzepte-Standards-Praxis, 4. Auflage, Heidelberg

Newman, S. (2015)
Building microservices: designing fine-grained systems, Beijing et al.

Newton-King, J. (2019)
gRPC vs HTTP APIs, https://devblogs.microsoft.com/aspnet/grpc-vs-http-apis/ (Zugriff am 15.08.2020)

Richards, M. (2016)
Microservices vs. service-oriented architecture, Beijing et al.

Rotem-Gal-Oz, A. (2006)
Fallacies of distributed computing explained, http://www. rgoarchitects. com/Files/fallacies. pdf (Zugriff am 15.08.2020)

The Linux Foundation (2020)
gRPC: A high-performance, open source universal RPC framework, https://grpc.io/ (Zugriff am 15.08.2020)

Torikian, G. et al. (2016)
The GitHub GraphQL API, https://github.blog/2016-09-14-the-github-graphql-api/ (Zugriff am 15.08.2020)

Van Steen, M.; Tanenbaum, A. S. (2018)
Distributed systems, Third edition, o.O.