derung der einzelnen Komponenten, eine entsprechende Rechenleistung genutzt werden. Des Weiteren gibt es auf Grund der Verteilung eine gewisse Ausfallsicherheit. Diese entsteht, da das Gesamtsystem nicht durch eine, sondern durch mehrere Maschinen getragen wird. Dadurch können einzelne Computer ausfallen ohne, dass das Anwendungssystem ausfällt. Jedoch kann dadurch ein Teil der gesamten Funktionalität wegfallen, der ggf. für das System notwendig ist. Um dies zu verhindern, sollten geeignete Maßnahmen getroffen werden, indem zum Beispiel nicht allein ein Computer für die Funktionalität verantwortlich ist, sondern mehrere.

Jedoch entsteht mit der Verteilung auch ein Anstieg der Komplexität, sowohl bei dem Konzipieren des Systems, als auch bei der Wartung und Managen dessen. Außerdem muss nicht nur ein Rechner abgesichert werden, sondern ein ganzes Netzwerk. Dadurch entsteht ein höherer Aufwand zur Absicherung.

Die einzelnen Computer können über verschiedene Mechaniken miteinander kommunizieren: Einerseits durch direkten Aufruf entfernter Funktionalität und andererseits durch indirekten Austausch von Informationen (vlg. Starke 2015, S. 116). Dabei kann der Transfer synchron oder asynchron ablaufen. Bei einem synchronen Aufruf, der nur direkt ausgelöst wird, führt ein Computer über das Netzwerk die Funktionalität eines anderen aus und wartet auf dessen Antwort. Bei einem asynchronen Aufruf wird entweder direkt oder indirekt Logik eines anderen Computer aufgerufen, während der aufrufende Rechner, ohne auf die Antwort zu warten, weiter verarbeitet. Ist die aufgerufene Rechner fertig gibt er das Resultat zurück, welches vom ersten Computer aufgenommen und verarbeitet wird.

Der Austausch von Informationen und das Aufrufen von externer Funktionalität kommt in einem System dauerhaft vor. Dadurch besteht das Risiko, dass einzelne Informationen verloren gehen können und das System sicherstellen muss, dass das Anwendungssystem nicht darunter leidet.

2.1.3 Interaktionsorientierte Systeme

Neben den verteilten Systemen gibt es noch weitere Architekturstile, zum Beispiel interaktionsorientierte Systeme. Sie zeichnen sich dadurch aus, dass sie den Fokus auf die Interaktion zwischen Mensch und Maschine legen (vlg. Starke 2015, S. 124). Ein viel verwendeter Vertreter hiervon ist der Model-View-Controller Ansatz. Hierbei werden die einzelnen Komponenten in drei unterschiedliche Kategorien eingeteilt, von dem jeweils ein Repräsentant vorhanden sein muss. Eingeteilt wird in die drei Kategorien: Model, View und Controller, unterdessen jede Gattung eine eigene Funktion besitzt. So kümmert sich das Model um die Datenspeicherung, den Datenabruf und Verarbeitung von Informationen. Davon getrennt sind die graphischen Darstellungen, welche durch Views definiert werden. Sie erhalten ihre Informationen vom Model. Das Verwalten der Benutzereinga-

be, sowie das weiterleiten zwischen einzelnen Views geschieht über den Controller. Er sorgt dafür, dass die Events oder Aktionen, die vom Benutzer über die Views auslöst werden, verarbeitet werden und führt entsprechende Datenverarbeitungen im Model aus. Abschließend updated er die erforderliche Darstellung.

Beim Model-View-Controller Ansatz handelt es sich um ein Muster, welches oft in der Softwarearchitektur verwendet wird. Im Unterschied zur vorhergehendem Stile stellt das Muster jedoch keine Anforderungen bezügliche der Hardware. Viel mehr beschreibt es eine Art den Quellcode hinsichtlich seiner Funktion zu teilen. Demnach kann dieser Stile auch für Codeteilung innerhalb einer Komponente verwendet werden und beschreibt nicht zwingend ein Architekturstile, der sich auf das Gesamtsystem besieht.

2.1.4 REST-Architektur

Neben den bislang genannten Architekturstilen gibt es eine Vielzahl von weiteren Strukturierungen, die sich erst in den letzten 20 Jahre entwickelt haben. Einer dieser Architekturstile ist die REST-Architektur, welche vom Miterfinder des HTTP-Standards Roy Fielding definiert wurde (Starke 2015, S. 128). Er beschrieb diesen Stile in seiner Dissertation an der Universität von Kalifornien im Jahr 2000 und charakterisiert ihn als Architekturstile fürs Web.

Dabei steht REST für *Representationl State Transfer*, welches ein Architekturstile für verteilte Systeme beschreibt und auf der Server-Client Architektur aufbaut (Fielding 2000, S. 76). Server-Client Architektur beschreibt eine Ausprägung eines verteilten Systems, bei dem die Anwendung in Server und Clients geteilt werden (Starke 2015, S. 117).[12]

Ein Server ist dabei eine Komponente im Netzwerk, welches Services anbietet. Ein Service könnte zum Beispiel sein, alle Informationen der hinterlegten Kunden auszugeben. Der Client hingegen konsumiert lediglich diese Informationen und dient dem Benutzer als Bedienungsoberfläche. Dies bedeutet, dass der Server nur passiv auf Anfragen vom Client wartet, während der Client selbst keine Informationen verarbeitet, sondern ausschließlich anzeigt.

Die REST-Architektur verwendet diese Aufteilung, um eine feste Trennung der Zuständigkeit zu integrieren (vgl. Fielding 2000, S. 78).

Die zweite Bedingung, die Fielding an den Architekturstile gestellt hat, ist dass die Kommunikation zustandslos, zu englisch (stateless), abläuft (Fielding 2000, S. 78). Dies bedeutet, dass die Nachrichten, die zwischen Server und Client ausgetauscht werden, alle nötigen Informationen beinhalten (Starke 2015, S. 128). Somit gibt der Server auf Anfrage des Clients stets die gleiche Antwort zurück, egal ob dieser zum ersten, oder wiederholten

[12]Hier kommt ein Text zur Einteilung des Begriffes Server-Client Architektur.

Mal angefragt wurde. Des Weiteren hängt die Antwort nicht vom Client ab. Diese Entkopplung zwischen den Komponenten ermöglicht, dass die Aufgabe des Server, sowie des Clients durch mehrere Computer verrichtet werden kann und somit das System skalierbar ist (Fielding 2000, S. 79).

Der Hauptunterschied zwischen der REST-Architektur und anderen Stilen liegt jedoch in der genauen Bestimmung der zu verwendeten Kommunikationsschnittstellen. So bestimmt die REST-Architektur sehr explizit, welche From zur Kommunikation verwendet werden darf. Anders als andere Stile beruht der Aufruf von Methodiken nicht auf individuelle Funktionalität, sondern auf dem HTTP-Standard. Konkret bedeutet dies, dass die einzelnen Dienste des Servers sich an die HTTP-Optionen (GET, PUT, POST und DELETE) richten und keinen eigenen verwenden (vlg. Starke 2015, S. 128). Somit baut die REST-Architektur auf ein Kommunikationsstandard auf, der sich im Internet etabliert hat.

Auf Grundlade der standardisierten Kommunikation können zwischen Server und Client intelligente Zwischenstationen geschalten werden, die zum Beispiel häufig vorkommende Anfragen abspeichern (vlg. Fielding 2000, S. 79 f. Starke 2015, S. 128). Somit lässt sich eine Vielzahl von Serveranfragen im vornherein beantworten.

Die Antwort des Servers erfolgt durch Repräsentationen der Daten, wovon es für jede Ressource mehr Formate gibt. So kann eine Schnittstelle abhängig des angeforderten Mediums, sowohl JSON, als auch XML oder HTML zurück geben (vgl. Starke 2015, S. 128).

Verwendet wird die REST-Architektur ausschließlich für Anwendungen im Internet, da es auf die Anwendung des Hypertext Transfer Protokoll (HTTP) angewiesen ist. Dabei findet der Architekturstile, sowohl Anwendung für ganze Systeme, als auch in komplexen Anwendungen mit einer Vielzahl ein einzelnen Services.

2.1.5 Einordnung des aktuellen Systems von PluraPolit

2.2 Microservices

Der Begriff Microservices wird mehrdeutig verwendet. So wird je nach Perspektive entweder eine Softwarearchitektur oder eine Komponente einer solchen Architektur beschrieben. Eng verbunden mit diesem Begriff sind auch dynamische Systeme und Konzeptionierung von Unternehmensstrukturen. Demnach sieht Eberhard Wolf unter Microservices ein Modellierungskonzept, welches dazu dient größere Softwaresysteme in kleinere Einheiten zu teilen (vgl. Wolff 2018, Kap. 1.1). Dabei hat die Aufteilung Auswirkungen auf die Organisation als auch auf die Entwicklungsprozesse.

Nach Sam Newman ist ein Microservice ein *„eigenständige ausführbare Softwarekompo-*