Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Ngoc Hien Le

Entwicklung einer Webanwendung zur Erstellung von individuellen Dashboards im Bereich Enterprise Architecture Management

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ngoc Hien Le

Entwicklung einer Webanwendung zur Erstellung von individuellen Dashboards im Bereich Enterprise Architecture Management

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Abgegeben am 04.09.2017

Ngoc Hien Le

Thema der Arbeit
Entwicklung einer Webanwendung zur Erstellung von individuellen Dashboards im Bereich Enterprise Architecture Management

Stichworte
Enterprise Architecture, ArchiMate, Visualisierung Analyse

Kurzzusammenfassung
In dieser Arbeit wurde eine Webanwendung entwickelt, die dem Benutzer individuelle Sicht auf die Unternehmensarchitektur bereitstellt. Die Unternehmensarchitekturdaten basieren auf ArchiMate, eine standardisierte Sprache für Enterprise Architecture Modellierung. Es wurden verschiedene Visualisierungsarten im Bereich Enterprise Architecture Management identifiziert und zwei für die prototypische Umsetzung ausgewählt.

Ngoc Hien Le

Title of the paper
Development of a web application for creating individual dashboards in the context of enterprise architecture management.

Keywords
Enterprise Architecture, ArchiMate, Visual Analysis

Abstract
In this bachelor thesis, a web application has been developed, which provide user individual views of the enterprise architecture. The enterprise architecture data are based on ArchiMate, a standard language for enterprise architecture modelling. Different visualization types haven been identified, from which two are chosen for the prototypical implementation.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Enterprise Architecture (EA) beschreibt das Zusammenspiel von Organisationsstruktur, Geschäftsprozessen, Informationssystemen und technischer Infrastruktur im Unternehmen. Modelle der Enterprise Architecture lassen sich mithilfe der Modellierungssprache ArchiMate erstellen und werden für eine ganzheitliche Kommunikation zwischen Stakeholdern verwendet. Verschiedene Stakeholder haben jedoch unterschiedliche Blickwinkel auf die Enterprise-Architecture-Daten. Je nach seiner Rolle und Aufgabe im Unternehmen verlangt jeder Stakeholder einen passenden Ausschnitt der Daten – in der für ihn am einfachsten zugänglichen Darstellungsform.

Um diese Anforderung zu erfüllen, werden verschiedene Visualisierungsarten verwendet. Anhand grafischer Darstellungen können Verantwortliche schnell Probleme erkennen und entsprechend handeln. Visualisierungen der Unternehmensarchitektur können je nach Bedarf des Benutzers in einem Dashboard zusammengefügt werden, um Beziehungen in der Architektur zu analysieren oder einen übergreifenden Blick auf die Enterprise-Architecture-Daten zu ermöglichen. Die Literatur und die softwaregestützte Methode in der Praxis fokussieren sich jedoch auf einzelne Visualisierungsarten und lassen somit die integrierte Darstellung der Unternehmensarchitektur offenbleiben.

1.2 Zielsetzung

Auf Grund der genannten Problemstellung werden in dieser Arbeit zwei Ziele gesetzt.

Zum einen sollen die relevanten Visualisierungsarten im Bereich Enterprise Architecture Management identifiziert und analysiert werden. Die Analyse soll auf dem aktuellen Stand von Literatur und Untersuchungen basierend. Dafür muss zuerst mit den Grundlagen von Enterprise Architecture beschäftigt werden.

Zum anderen soll eine Webanwendung entwickelt werde, die die Erstellung von zwei typischen ausgewählten Visualisierungsarten ermöglicht. Daten zur Visualisierung der Unternehmensarchitektur basieren auf der ArchiMate Sprache. Die Anwendung bietet Benutzern zudem auch die Möglichkeit, verschiedene Visualisierungen individuell in einem Dashboard zusammenzufügen. Die technische Strukturierung der Anwendung soll klar definiert werden.

1.3 Aufbau der Arbeit

Nach einer kurzen Einführung in das Thema im Kapitel 1 werden die Grundlagen von Enterprise Architecture im Kapitel 2 vorgestellt. Auf die Modellierung von Unternehmensarchitektur mit der Sprache ArchiMate und deren Grundkonzepte wird auch eingegangen. Danach befasst sich die Arbeit mit den Visualisierungsmöglichkeiten von Unternehmensarchitektur.

Das dritte Kapitel stellt zunächst ein Metamodell vor, mit dem ArchiMate Daten gespeichert werden. Anschließen sind verschiedene Visualisierungsarten im EAM-Bereich genauer zu betrachten. Die funktionale sowie nicht-funktionale Anforderungen an die Anwendung werden im Detail beschrieben.

Aufbauend auf Kapitel 3 entsteht im Kapitel 4 zum einen die grafische Benutzeroberfläche der Anwendung, zum anderen wird die Architektur zusammen mit allen Komponenten und Schnittstellen dargestellt. Auch der algorithmische Ansatz zur Visualisierung von ArchiMate-Daten lässt sich in diesem Kapitel vorstellen.

Die technische Realisierung der Anwendung wird im Kapitel 5 anhand eines Szenarios veranschaulicht. Das letzte Kapitel fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick über Weiterentwicklungsmöglichkeiten wieder.

2 Grundlagen

2.1 Enterprise Architecture

2.1.1 Begrifflichkeiten

Systeme haben Architekturen. Der Standard ISO/IEC/IEEE 42010 [1] gibt eine Definition von Architektur eines Systems an:

***Architecture**: fundamental concepts or properties of a system in its environment, embodied in its elements, relationships, and in the principles of its design and evolution.*

Der Begriff *System* lässt sich allerdings in dem Standard ISO/IEC/IEEE 42010 nicht explizit definieren, sondern wird als ein Platzhalter für unterschiedliche ‚Dinge' genutzt, wie z. B. *software products and services* oder *software-intensive systems* [1]. Damit kann der Systembegriff auch auf Unternehmen (engl.: Enterprise) angewendet werden. *Enterprise* versteht sich gemäß dem Open Group Standard als eine Sammlung von Organisationseinheiten, die gemeinsame Ziele haben [2]:

***Enterprise:** any collection of organisations that has a common set of goals and/or a single bottom line.*

In diesem Sinne umfasst ein Enterprise die kompletten soziologischen und technischen Systeme einer Organisation, inklusive deren Menschen, Informationen, Prozessen, Technologien [3] und beschränkt sich nicht nur auf betriebliche Unternehmen, sondern wird z. B. auch von Institutionen oder dem Militär benutzt.

Ein Enterprise setzt sich aus verschiedenen Systemen mit unterschiedlichen Architekturen zusammen. Die Gesamtheit aller Architekturen eines Unternehmens wird in [4] als „Enterprise Architecture" (Unternehmensarchitektur) betrachtet. Dementsprechend definieren auch Lankhorst et al. in [5] den Begriff *Enterprise Architecture*:

***Enterprise architecture:** a coherent whole of principles, methods, and models that are used in the design and realisation of an enterprise's organisational structure, business processes, information systems, and infrastructure.*

In der Literatur wird sich auch darauf geeinigt, dass Enterprise Architecture unterschiedliche Architekturbereiche bzw. Architekturebenen eines Unternehmens betrachtet (vgl. [4], [5], [6]). Die sogenannte Architekturpyramide (Abbildung 1) von Keller [7] liefert z. B. eine mögliche Aufteilung der Unternehmensarchitektur in verschiedene Ebenen: Strategie, Geschäftsarchitektur, Informationsarchitektur, IT-Architektur und IT-Basisinfrastruktur.

Abbildung 1: Unternehmensarchitekturpyramide nach Keller [7]

Zu beachten ist allerdings, dass Enterprise Architecture eher eine Black-Box-Sicht auf die einzelnen Architekturelemente und deren Beziehungen darstellt. Die interne White-Box-Sicht auf die Elemente ist weiterhin Gegenstand der einzelnen Architekturdisziplinen, wie z. B. Softwarearchitektur [4].

2.1.2 Der Nutzen von Enterprise Architecture

Es gibt unterschiedliche Faktoren, die den zunehmenden Einsatz von Enterprise Architecture in Unternehmen fördern, wie z. B. Änderungen am Markt, regulatorische Anforderungen, organisatorische Strukturänderung oder technologische Innovation. All diese Faktoren haben Auswirkungen auf alle Ebenen der Unternehmensarchitektur. Änderungen der Geschäftsarchitektur etwa müssen sich in der IT-Architektur widerspiegeln. Hierfür reicht ein gutes Verständnis einzelner Ebenen oft nicht aus, sondern es spielen auch die ebenenübergreifenden Beziehungen zwischen den Architekturelementen eine wichtige Rolle. Lankhorst et al. weisen darauf hin, dass eine Optimierung innerhalb einer Architekturebene nicht notwendigerweise zu einer gesamtheitlichen Optimierung der Unternehmensarchitektur führt [5]. Mit Enterprise Architecture wird das Zusammenspiel von Geschäftsprozessen, Anwendungssystemen und Technologien in Unternehmen klar dokumentiert, Informationen aus unterschiedlichen Domänen werden verknüpft. Durch diesen ganzheitlichen Blick lässt sich Transparenz schaffen. Entscheidende Fragen, z. B. wie

ein Unternehmen organisatorisch strukturiert ist oder wie die geschäftlichen Tätigkeiten des Unternehmens von der Informationstechnologie (IT) unterstützt werden, können schnell beantwortet werden.

Jedoch könnten sich aus der Entwicklung von Enterprise Architecture unerwartete Ergebnisse ergeben, wie Ahlemann et al. in [6] beschreiben: Unternehmen reagieren auf die stetig neuen und steigenden Anforderungen mit Restrukturierung der Geschäftsprozesse und somit auch der darunterliegenden IT-Systeme. Diese Änderungen der Unternehmensarchitektur dienen zwar ursprünglich der Steigerung der Konkurrenzfähigkeit des Unternehmens, könnten aber zu Heterogenität und höherer Komplexität führen, wenn die Änderungen einzeln durchgeführt werden, ohne dass es eine unternehmensübergreifende Koordination dafür gibt.

Aus diesem Grund bietet das Enterprise Architecture Management (EAM, dt. Unternehmensarchitekturmanagement) eine Reihe von Richtlinien und Prinzipien, die der Steuerung und Planung der Unternehmensarchitektur dienen. Mit dem Einsatz von Enterprise Architecture Management sollen die Strategie und geschäftlichen Anforderungen des Unternehmens **effektiv und effizient** durch die IT unterstützt werden [4].

2.2 Enterprise Architecture Modellierung

Der Standard ISO/IEC/IEEE 42010 [1] definiert *Architecture Description* als Beschreibungsmittel für eine Architektur. Die Form einer Architekturbeschreibung wird vom ISO/IEC/IEEE 42010 Standard nicht vorgeschrieben, jedoch in der Praxis als Modelle mithilfe von Modellierungssprachen erstellt. In den Teilbereichen der Enterprise Architecture gibt es bereits viele verbreitete Modellierungssprachen und unterstützende Tools, wie z. B. *BPMN*[1] für die Modellierung von Geschäftsprozessen oder *UML*[2] im Bereich von Software-Engineering. Die einzelnen Modelle aus den verschiedenen Architekturebenen, die mit unterschiedlichen Modellierungssprachen erstellt werden, eignen sich allerdings nicht als Beschreibungsmittel für die Unternehmensarchitektur eines Unternehmens. Ein Grund dafür ist, dass diese keine Zusammenhänge zwischen den Architekturebenen darstellen und eventuell Inkonsistenz zur Folge haben können [5].

2.2.1 Das ArchiMate Core Framework

Als Lösung für eine einheitliche und standardisierte Modellierung der Unternehmensarchitektur bietet sich die Modellierungssprache **ArchiMate®** von The Open Group an [8]. Damit sollen Beschreibung, Analyse und unternehmensübergreifende

[1] http://www.omg.org/spec/BPMN/ Zugriff am [31.05.2017]
[2] http://www.omg.org/spec/UML/ Zugriff [31.05.2017]

Kommunikation von Enterprise Architecture erleichtert werden. Aktuell befindet sich die Sprache ArchiMate in der Version 3.0. In dieser Arbeit wird sich nur mit dem ArchiMate Core Framework beschäftigt, was den Kern der Sprache auszeichnet [8].

ArchiMate bietet neben einer Menge von Entitäten und Beziehungen zur semantischen Beschreibung auch grafische Notation zur symbolischen Beschreibung einer Enterprise Architecture. Der Begriff *Element* bildet die Basiseinheit in ArchiMate und lässt sich vertikal in drei unterschiedliche Ebenen (engl. Layer) aufteilen:

Business Layer: beinhaltet geschäftliche Elemente wie z. B. Produkte, Dienstleistungen etc.

Application Layer: stellt Softwareanwendungen und deren Anwendungsservice dar, die für die Unterstützung von Business Layer genutzt werden.

Technology Layer: umfasst technische Infrastruktur-Dienste, die für die Ausführung von Anwendungen in Application Layer benötigt werden.

Zudem können Elemente unabhängig von den zugehörigen Ebenen horizontal nach drei Aspekten klassifiziert werden:

Active Structure Aspect: repräsentiert strukturelle Elemente, die ein Verhalten ausüben können.

Behavior Aspect: stellt das Verhalten dar, das von strukturellen Elementen ausgeübt wird.

Passive Structure Aspect: repräsentiert Objekte, auf die ein Verhalten ausgeübt werden kann.

Die drei Aspekte in Archimate lehnen sich an die natürliche Sprache an, wobei ein Satz aus *Active Structure Element* als Subjekt, *Behavior* als Verb und *Passive Structure Element* als Objekt besteht.

Abbildung 2 zeigt eine grafische Übersicht des ArchiMate Core Frameworks.

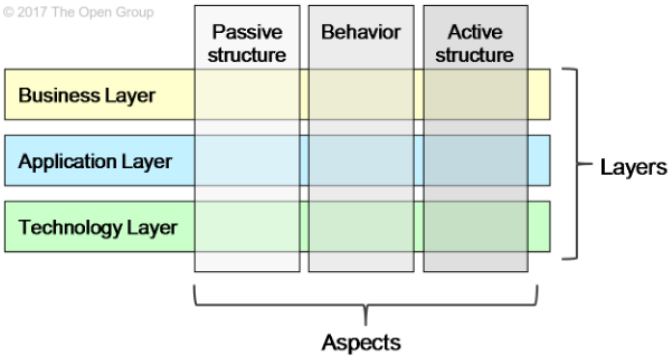


Abbildung 2: Das ArchiMate Core Framework [8]

2.2.2 Beziehungen in ArchiMate

Für die Beschreibung von Beziehungen zwischen Elementen wird in ArchiMate das Konzept *Relationship* benutzt. Ein Relationship wird als eine Verbindung zwischen einem Quell- und einem Zielelement definiert. ArchiMate unterscheidet dabei vier unterschiedliche Gruppen: *Structural Relationships* (strukturelle Beziehungen), *Dependency Relationships* (Abhängigkeitsbeziehungen), *Dynamic Relationships* (dynamische Beziehungen) und *Other Relationships* (andere Beziehungen). In Abbildung 3 werden die einzelnen Beziehungstypen in ArchiMate und deren symbolische Notation aufgelistet.

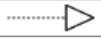
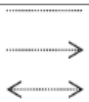
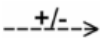
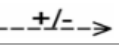
Structural	**Notation**
Composition	
Aggregation	
Assignment	
Realization	
Dependency	**Notation**
Serving	
Access	
Influence	
Dynamic	**Notation**
Triggering	
Flow	
Other	**Notation**
Specialization	
Association	
Junction	 (And) Junction Or Junction

Abbildung 3: Relationships in ArchiMate [8]

Das generische Metamodell in ArchiMate umfasst alle Elemente, horizontal klassifiziert nach drei Aspekten wie oben beschrieben, und die Beziehungen zwischen diesen Elementen.

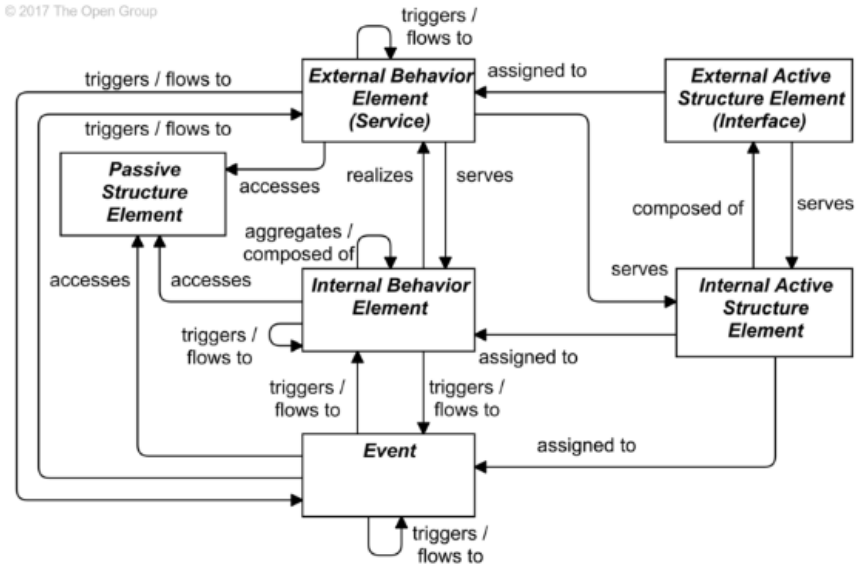


Abbildung 4: Das ArchiMate Metamodell [8]

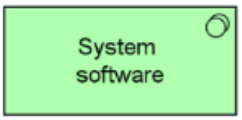
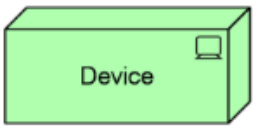
In Abbildung 4 werden nicht alle erlaubten Beziehungen angezeigt. Zwischen zwei beliebigen Elementen in ArchiMate kann es entweder eine direkte Beziehung oder eine indirekte Beziehung geben. In der ArchiMate-Spezifikation werden Regeln für Ableitungen solcher indirekten Beziehungen beschrieben. Es ist nicht immer sinnvoll und nötig, eine direkte Beziehung zwischen zwei Elementen zu setzen. Mit den Ableitungsregeln wird es möglich, die irrelevanten Zwischenelemente eines Modells auszublenden und somit eine Impact-Analyse zu erleichtern [8].

2.2.3 Elementtypen in ArchiMate

Die Elementtypen innerhalb einzelner vertikaler Ebenen der Unternehmensarchitektur werden auch in ArchiMate spezifiziert [8]. Im Folgenden werden nur die für diese Arbeit relevanten Konzepte detailliert aufgezeigt.

Tabelle 1: Einige Elementtypen in ArchiMate

Business Layer		Notation
Business Actor	Ein Akteur beschreibt eine Business-Entität, die in der Lage ist, Verhalten umzusetzen. Beispiele sind Menschen, Abteilungen und Geschäftseinheiten.	

Business Function	Eine Geschäftsfunktion ist eine Ansammlung von Geschäftsverhalten, basierend auf einen gewählten Kriteriensatz (z. B. Skills, Leistungen, Ressourcen). Es gibt eine potenzielle n:n-Beziehung zwischen Geschäftsprozessen und Funktionen.	
Business Process	Ein Geschäftsprozess repräsentiert eine Abfolge von geschäftlichen Verhalten, die ein bestimmtes Ergebnis wie etwa Produkte oder Services erreicht.	
Application Layer		
Application Component	Eine Applikationskomponente repräsentiert eine modulare und ersetzbare Kapselung der Anwendungsfunktionalitäten.	
Technology Layer		
System Software	Systemsoftware repräsentiert Software, die eine Umgebung für Speichern, Ausführen und Verwenden von Applikationskomponenten bereitstellt. Beispiele sind Betriebssysteme, Datenbanksysteme oder Anwendungsserver.	
Device	Ein Device ist eine physische IT-Ressource, auf der Systemsoftware oder Artefakte gespeichert oder ausgeführt werden.	

Die Basiselemente und Beziehungen in ArchiMate dienen nur dem allgemeinen Zweck der Enterprise Architecture Modellierung. Ein Erweiterungsmechanismus wird in ArchiMate auch unterstützt, um domänenspezifizierte Modellierung zu ermöglichen [8]. Mittels eines *Profile* werden jedem Element und jeder Beziehung in ArchiMate beliebig viele typisierte Attributen hinzugefügt. So kann beispielsweise einem Application Component das Attribut *Wartungskosten* vom Typ *Währung* mit dem Wert *5000* zugeordnet werden.

The Open Group bietet neben der ArchiMate-Spezifikation noch ein standardisiertes Dateiformat zum Austauschen von ArchiMate-Modellen zwischen unterschiedlichen Tools. Es wird aber darauf hingewiesen, dass dieses Dateiformat nicht zum dauerhaften Speichern der ArchiMate-Daten verwendet werden soll [9].

2.3 Enterprise Architecture Visualisierung

Modelle dienen als ganzheitliches Beschreibungsmittel der Unternehmensarchitektur. Doch entsteht ein realer Nutzen erst dann, wenn die Architekturbeschreibungen adäquat und zielgruppengerecht aufbereitet werden [10]. Elemente der Unternehmensarchitektur und deren Zusammenhänge lassen sich systematisch aus verschiedenen Blickwinkeln visualisieren, um Fragestellungen verschiedener Stakeholder zu analysieren und zu beantworten.

2.3.1 Visualisierung mit ArchiMate

Visualisierungen von Enterprise Architecture können gemäß dem Standard ISO/IEC/IEEE 42010 [1] mit *Views* (Sicht) und *Viewpoints* (Perspektive) strukturiert werden. Eine View ist eine Repräsentation eines Systems aus der Perspektive eines Stakeholders. Die Konventionen zur Erstellung und Nutzung einer View wird von einem Viewpoint spezifiziert. Eine View beschreibt das, was gesehen wird, und ein Viewpoint bestimmt, aus welcher Perspektive es gesehen wird [5].

ArchiMate verfolgt auch diesen Ansatz und bietet einen Mechanismus zur Erstellung von Views und Viewpoints, mit denen Interessen verschiedener Stakeholder berücksichtigt werden können [8]. Tabelle 2 beschreibt die Schritte zur Erstellung eines Viewpoints in ArchiMate.

Tabelle 2: Erstellung eines Viewpoints in ArchiMate

1.	Relevante Konzepte (Elemente und Beziehungen) des ArchiMate-Metamodells auswählen, basierend auf Fragestellungen und Interessen eines bestimmten Stakeholders.
2.	Eine passende Darstellung der ausgewählten Konzepte definieren, die für den Stakeholder einfach zu verstehen ist. Es können die Standard-Notation von ArchiMate oder andere Visualisierungsarten verwendet werden.

In der ArchiMate-Spezifikation [8] sind einige Viewpoints vordefiniert und klassifiziert. Abbildung 5 zeigt ein Beispiel des *Layered Viewpoint* und dessen Visualisierung mit der Standard-Notation von ArchiMate. Die Beispieldaten stammen aus der Fallstudie *ArchiSurance* [11] der Open Group.

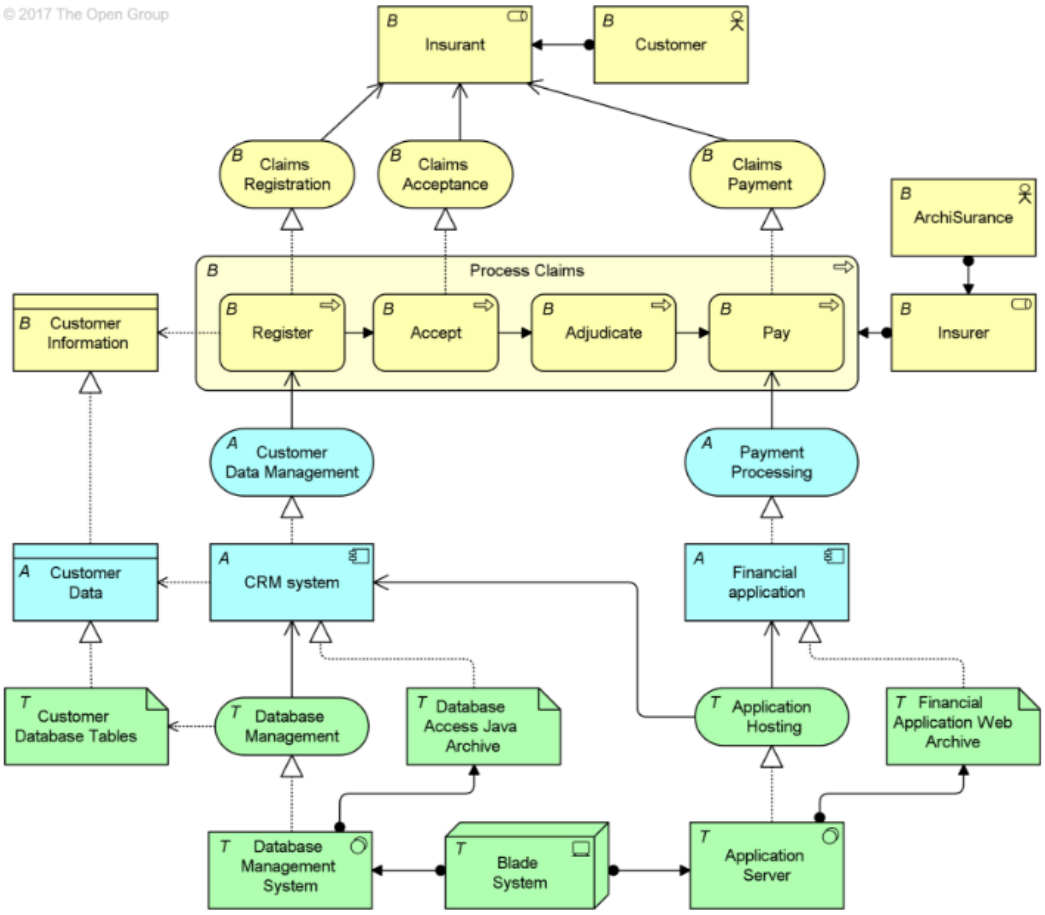


Abbildung 5: Beispiel Lauere Viewpoint und Visualisierung mit der Standard-Notation von ArchiMate [8]

Die Visualisierung der Unternehmensarchitektur mit ArchiMate-Notation setzt allerdings ein gutes Verständnis der Sprache voraus, was die Kommunikation zwischen technischen und nicht-technischen Stakeholdern erschweren könnte.

2.3.2 Visualisierung in der Literatur und Praxis

In der Literatur beschäftigt sich die Softwarekartografie auch mit der Visualisierung von Enterprise Architecture. Wittenburg [12] identifiziert verschiedene Typen von Softwarekarten, wie z. B.:

Clusterkarte: fasst die Elemente in logischen, ggf. geschachtelten Domänen zusammen, beispielsweise Funktionsbereiche oder Organisationsbereiche.

Kartesische Karte, z. B.:

Prozessunterstützungskarte: stellt dar, welche ‚Abteilung' welche ‚Anwendung' für welchen ‚Geschäftsprozess' verwendet.

Zeitintervallkarte: stellt Zeitverläufe dar, z. B. Projektverläufe.

Entsprechend dieser Kategorisierung schlägt Hanschke [10] eine umfassende Liste von Best-Practice-Visualisierungen der Unternehmensarchitektur vor, z. B. Clustergrafik, Bebauungsplangrafik, Informationsflussgrafik. Abbildung 6 gibt eine Übersicht davon.

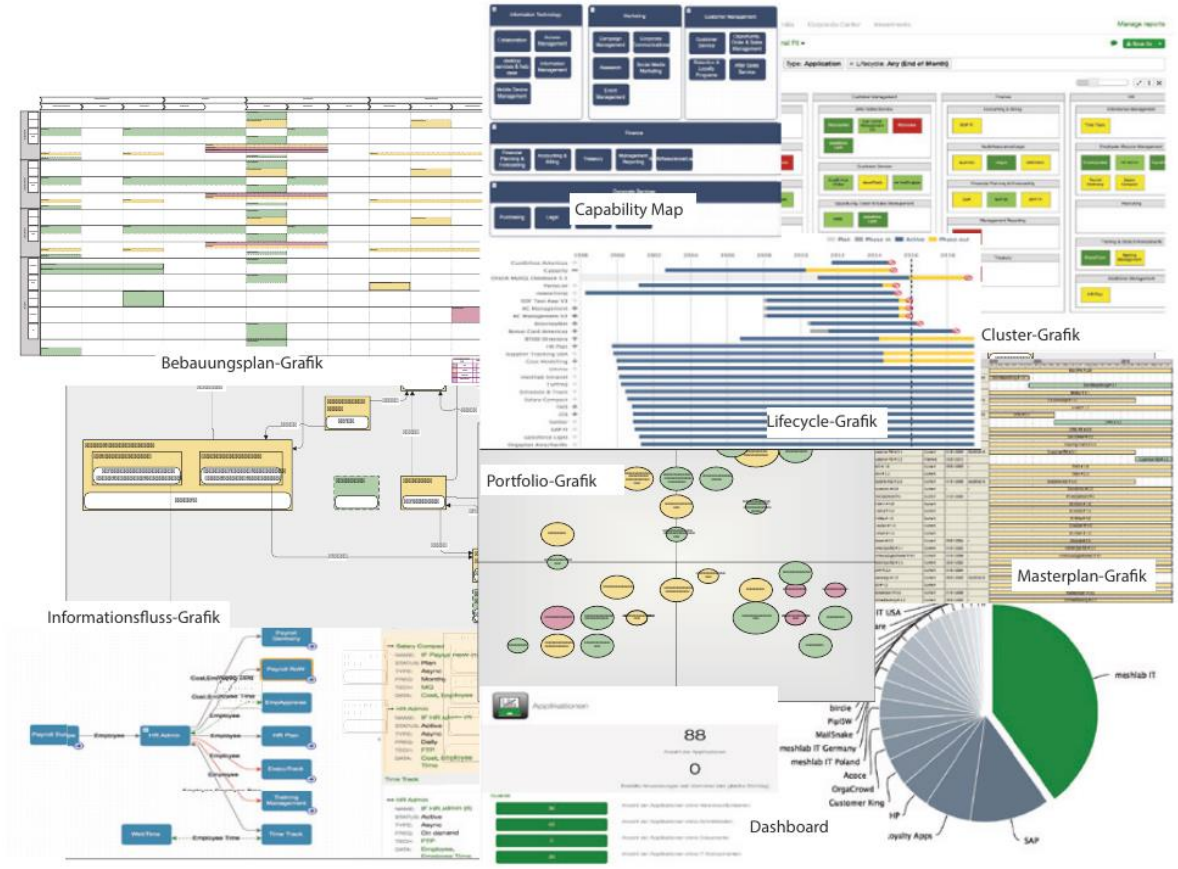


Abbildung 6: Best-Practice-Visualisierungen nach Hanschke [10]

Die softwaregestützte Visualisierung der Enterprise Architecture ist auch ein Forschungsgegenstand im EAM-Bereich. Kruse et al. diskutieren in [13] die Motivation und Vorteile der Entkopplung von Modellen und Visualisierungen der Unternehmensarchitektur. Mit der Technik von Modelltransformationen und *high-order*-Transformationen stellen Kruse et al. einen Prototyp vor, mit dem sich Visualisierungen erstellen und konfigurieren lassen, ohne dass es zu vieler Abhängigkeiten mit dem darunterliegenden Modell bedarf. Die

prototypische Implementierung basiert auf dem Eclipse Modeling Framework[3] und nutzt das *Matrix-map* als Demonstration. Allerdings erkennen Kruse et al. auch, dass dieser Ansatz noch wissensintensiv und fehleranfällig ist.

Mit dem Fokus auf Business-User schlagen Roth et al. [14] einen anderen Ansatz zur benutzerfreundlichen Konfiguration von Visualisierungen vor. Hierfür stellen die Autoren ein Metamodell vor, das sowohl für die Unternehmensarchitektur als auch für verschiedene Visualisierungsarten benutzt wird. Des Weiteren wird ein struktureller *pattern matching*-Algorithmus entwickelt, um die Visualisierung mit dem Architekturmodell zu verknüpfen. Den nicht-technischen Benutzern stellen Roth et al. ein prototypisches Werkzeug zur Erstellung und Konfiguration verschiedener Visualisierungsarten wie Bubble Chart, Cluster Map, Gantt Chart, Ternary Matrix etc. zur Verfügung.

Jedoch fokussieren sich die Prototypen in [13] und [14] nur auf Visualisierungsarten, die Beziehungen zwischen Elementen der Unternehmensarchitektur darstellen. Die Architekturelemente können aber auch individuelle Attribute haben, wie z. B. Wartungskosten von *Application Component* oder technischer Zustand von *Device*. Monahov [15] begründet, dass Kennzahlen erforderlich sind, um das EAM effektiv zu unterstützen. In diesem Sinne schlägt Schätzlein [16] fünf Diagrammtypen zur Visualisierung von EAM-Kennzahlen vor: Liniendiagramm, Spaltendiagramm, Kreisdiagramm, Kiviat-Diagramm und Bullet-Diagramm. Die Kennzahlen basieren auf dem EAM KPI Katalog der TU München [17].

Zahlreiche Visualisierungsarten der Unternehmensarchitektur werden in kommerziellen Softwares und Open-Source-Softwares unterstützt – in unterschiedlichem Umfang und mit unterschiedlicher Flexibilität. In [18] wurden 26 Visualisierungsarten identifiziert und 19 Enterprise-Architecture-Softwares nach Visualisierungs- und Visualisierungskonfigurationsmöglichkeit detailliert untersucht. Es wurden 109 Unternehmensarchitekten befragt. Es ergab sich daraus, dass die Visualisierungsarten zur Darstellung von Beziehungen (Matrix, Clustergrafik etc.) am häufigsten verwendet werden. Laut der Untersuchung scheint die Visualisierung von quantitativen Daten (Kennzahlen) weniger bedeutend im EAM, was möglicherweise auf eine niedrige Datenqualität zurückzuführen sein könnte.

[3] http://www.eclipse.org/modeling/emf/. Zugriff am [31.05.2017]

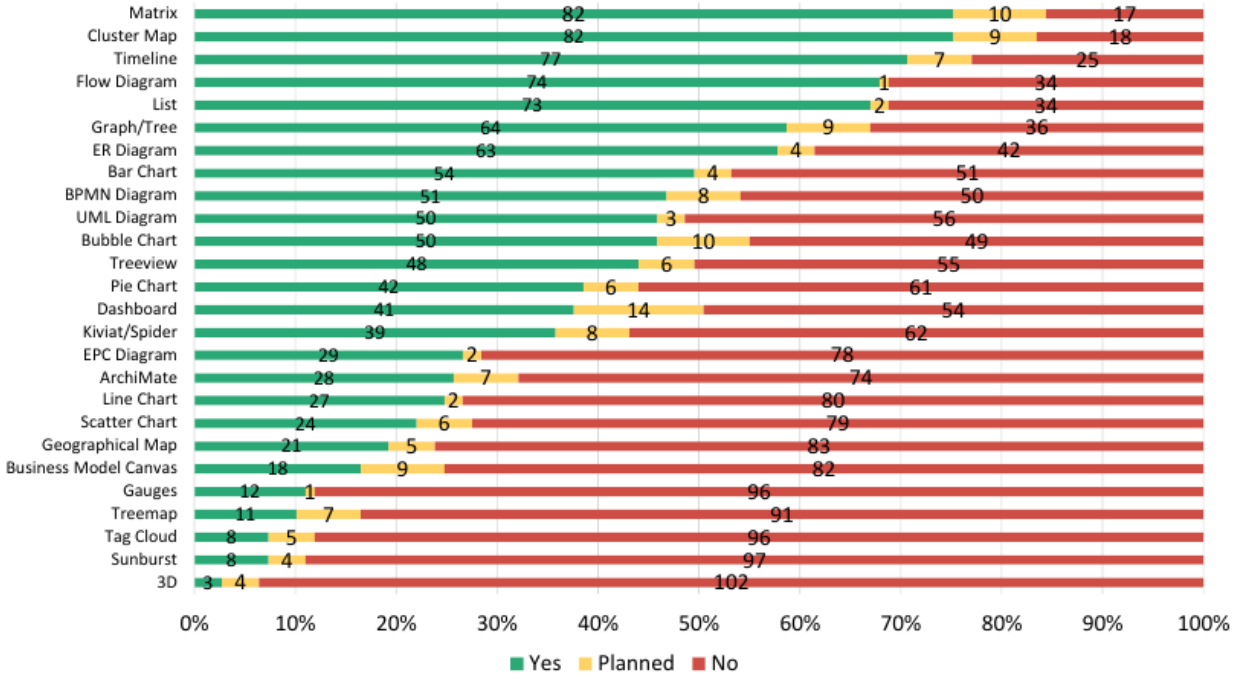


Abbildung 7: Verwendung von Visualisierungsarten [18]

3 Analyse

Aufbauend auf den in Kapitel 2 dargestellten Grundlagen werden in diesem Kapitel diverse Analysen durchgeführt, um die Rahmenbedingung und Anforderungen an die zu entwickelnde Anwendung zu identifizieren. Zunächst wird ein ArchiMate-konformes Metamodell für die Unternehmensarchitektur vorgestellt, folgend die Darstellung der Entscheidung über die zu unterstützenden Visualisierungsarten in der Anwendung. Die funktionalen und nicht-funktionalen Anforderungen der Anwendung werden abschließend im Detail beschrieben.

3.1 Das EA-Metamodell

Basierend auf dem Konzept und dem Metamodell der ArchiMate-Sprache wird ein vereinfachtes EA-Metamodell vorgeschlagen, das in dieser Arbeit als Datenmodell zur Speicherung der ArchiMate-Daten und als Objektmodell zur objektorientierten Implementierung des Prototyps verwendet werden kann.

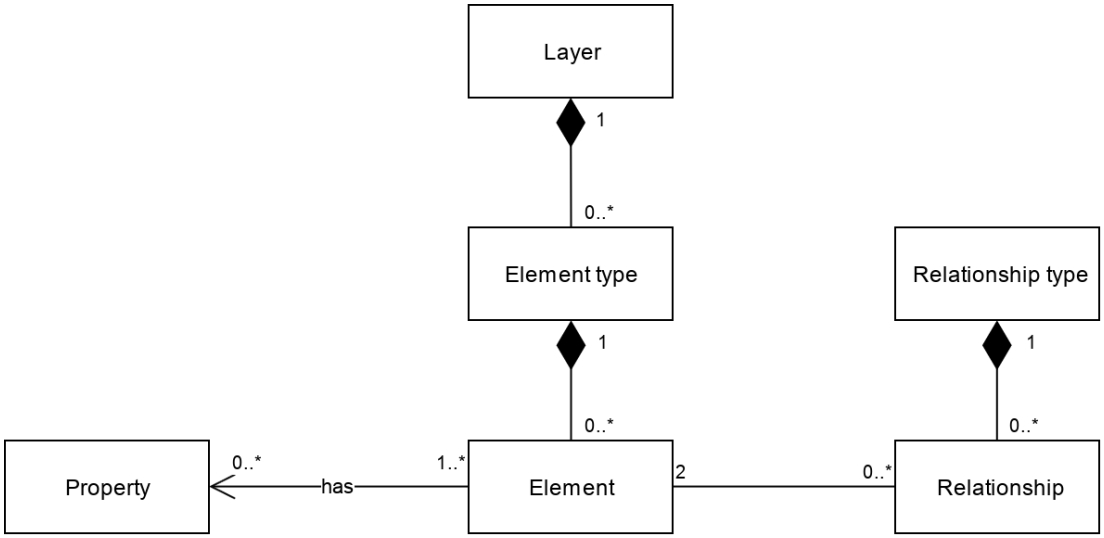


Abbildung 8: Das EA-Metamodell

Layer stellt die Architekturebene dar, wie z. B. Business Layer oder Application Layer.

Element type repräsentiert die konzeptionellen Elemente der Unternehmensarchitektur. Jeder Element type gehört zu genau einem Layer, z. B. Business Process gehört zum Business Layer oder Application Component zum Application Layer.

Element stellt die Instanzen eines Element Types dar, z. B. ist *CRM System* eine Instanz von Application Component.

Relationship type repräsentiert die verschiedenen Beziehungstypen in der Unternehmensarchitektur, z. B. Composition, Realization oder Serving.

Relationship instanziiert Relationship type und stellt eine direkte, gerichtete Beziehung zwischen zwei Elementen dar. Eine Beziehung ist immer genau einem Quell- und einem Zielelement zugeordnet, z. B. *CRM System* realisiert *Customer Data Management*.

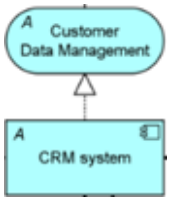


Abbildung 9: Relation eines EA-Metamodells [8] – Beispiel

Property stellt die Attribute dar, die den Elementen zugeordnet werden können, z. B. das Attribut *health status* kann einer Application Component, einer System Software oder einem Device zugeordnet werden. Property entspricht dem Konzept von Profile in der ArchiMate-Sprache.

Für die prototypische Implementierung in dieser Arbeit wird auch das Attribut *health status* genutzt, um den technischen Zustand von betrieblichen Anwendungen, Systemanwendungen und Geräten darzustellen. Erlaubte Werte für *health status* sind *good, average, bad*.

Mit dem vorgeschlagenen EA-Metamodell lassen sich alle Kernkonzepte und der Erweiterungsmechanismus von ArchiMate abbilden.

3.2 Die Visualisierungsarten

Wie bereits in 2.3.2 beschrieben, sind im EAM-Bereich sowohl Visualisierungen von Beziehungen als auch Visualisierungen von Kennzahlen von Nutzen. In folgendem Abschnitt werden zwei Visualisierungsarten vorgestellt, zum einen für die Darstellung von Beziehungen und zum anderen für die Darstellung quantitativer Daten der Unternehmensarchitektur.

3.2.1 Untersuchungsergebnis

Verschiedene Visualisierungsarten und deren Verwendungsintensität wurden von der TU München untersucht (vgl. Abbildung 7) [18]. Das Ergebnis wird in Tabelle 3 um das Merkmal ‚Visualisierungszweck' erweitert und nochmals veranschaulicht.

Tabelle 3: Visualisierungsarten und deren Verwendungsintensität nach [18]

Visualisierungsart	Verwendungsintensität	Visualisierungszweck
Matrix	75%	Beziehungen
Cluster Map	75%	Beziehungen
Timeline	70%	Beziehungen
Flow Diagram	67%	Beziehungen
List	66%	Beziehungen
Graph/Tree	58%	Beziehungen
ER Diagram	57%	Beziehungen
Bar Chart	49%	Kennzahlen
BPMN Diagram	46%	Beziehungen
UML Diagram	45%	Beziehungen
Bubble Chart	45%	Beziehungen und Kennzahlen
Treeview	44%	Beziehungen
Pie Chart	38%	Kennzahlen
Dashboard	37%	Kennzahlen
Kiviat/Spider	35%	Kennzahlen
EPC Diagram	26%	Beziehungen
ArchiMate	25%	Beziehungen und Kennzahlen
Line Chart	24%	Kennzahlen
Scatter Chart	22%	Kennzahlen
Geographical Map	19%	Beziehungen
Business Model Canvas	16%	Beziehungen
Gauges	11%	Kennzahlen
Treemap	10%	Beziehungen und Kennzahlen
Tag Cloud	7%	Kennzahlen
Sunburst	7%	Kennzahlen
3D	2%	Beziehungen und Kennzahlen

Das Merkmal ‚Visualisierungszweck' stellt jedoch nur eine relative Einschätzung dar, welche Visualisierungsart typischerweise für welchen Zweck genutzt wird. Eine klare Abtrennung zwischen beiden Zweckgruppen ist nicht Bestandteil dieser Arbeit.

Aus Tabelle 3 lässt sich feststellen, dass *Matrix* für die Visualisierung von Beziehungen und *Bar Chart* für die Visualisierung von Kennzahlen am häufigsten verwendet werden. Diese beiden Visualisierungsarten werden im Folgenden detailliert betrachtet.

3.2.2 Bebauungsplangrafik

Eine Bebauungsplangrafik, auch Matrix-Diagramm genannt [10], ermöglicht die Darstellung einer ternären Beziehung in Form einer Matrix. In [5] illustrieren Lankhorst et al. auch die Bebauungsplangrafik mit dem englischen Begriff *Landscape Diagram*. Nach Wittenburg [12] lässt sich eine Bebauungsplangrafik der Kategorie *Kartesische Softwarekarte* zuordnen. Die Prozessunterstützungskarte (vgl. 2.3.2) repräsentiert eine spezifische Form der Bebauungsplangrafik.

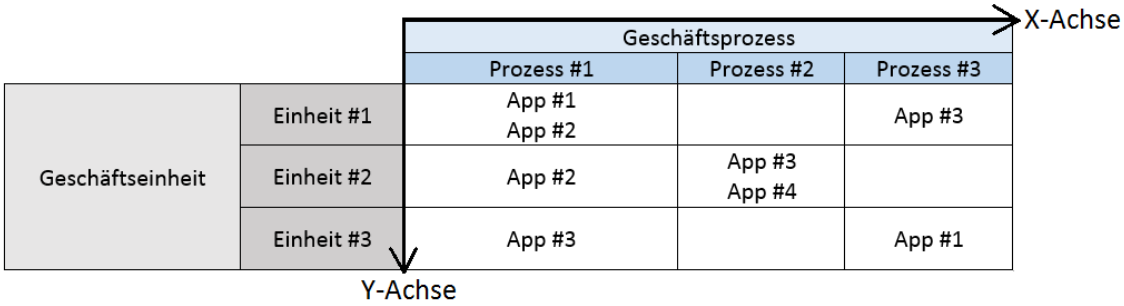


Abbildung 10: Beispiel Bebauungsplangrafik

Abbildung 10 zeigt beispielhaft die Beziehung zwischen Geschäftsprozessen, Anwendungen und Geschäftseinheiten. Hier werden die Geschäftsprozesse auf der X-Achse, die Geschäftseinheiten auf der Y-Achse und die Anwendungen in der Zelle verortet. Daraus lässt sich erkennen, welche Geschäftseinheit welche Anwendungen für welchen Geschäftsprozess verwendet. Bemerkenswert ist auch, dass jedem Paar Geschäftsprozess-Geschäftseinheit keine oder mehrere Anwendungen zugeordnet werden können. Zudem zeigt Hanschke [10], dass zusätzliche Informationen durch die Verwendung unterschiedlicher Farben und Linientypen bereitgestellt werden können. So kann z. B. die Anwendung *App #1* in Abbildung 10 mit der Farbe Grün markiert werden, um den technischen Zustand *good* (vgl. 3.2.1) darzustellen.

Die Dimensionen (X-Achse, Y-Achse und Zellen) der Bebauungsplangrafik können beliebig aus den Elementen der Unternehmensarchitektur ausgewählt werden, um eine Vielzahl von Fragestellungen der Form „Wer nutzt was wofür“ zu beantworten. Zudem ist die Beziehung zwischen den Elementen in Achsen und Zellen nicht notwendigerweise direkt in dem Architekturmodell abgebildet, sondern kann nach bestimmten Regeln abgeleitet werden. Hanschke [10] unterscheidet drei Arten von Bebauungsplangrafiken.

Fachliche Bebauungsplangrafik: stellt die fachlichen Abhängigkeiten zwischen Elementen im Business Layer dar. Beispielsweise werden Geschäftsprozesse und Geschäftseinheiten auf den Achsen und die fachlichen Funktionen in den Zellen verortet.

Technische Bebauungsplangrafik: stellt die technischen Abhängigkeiten zwischen Elementen im Application Layer und/oder dem Technology Layer dar, beispielsweise Anwendungen und Geräte als Achsen und Systemsoftwares in den Zellen. Damit lässt sich z. B. sehen, dass die Anwendung *CRM System* die Datenbank *Oracle DBMS* nutzt, die auf dem *Unix Server* gehostet werden.

‚Typische' Bebauungsplangrafik: stellt die IT-Unterstützung des Geschäfts dar, beispielshaft dargestellt in Abbildung 10.

3.2.3 Balkendiagramm

Ein Balkendiagramm visualisiert aggregierte Daten bezüglich verschiedener Kategorien. Jede Kategorie wird als ein waagerecht liegender Balken dargestellt, dessen Länge proportional zum repräsentierten Wert ist. Das Balkendiagramm ist geeignet für den direkten Vergleich der Daten. Abbildung 11 vergleicht beispielsweise die Anzahl von Elementen bezüglich deren technischen Zustand.

Abbildung 11: Beispiel Balkendiagramm

3.3 Funktionale Anforderungen

Funktionalitäten, die einem Anwender die Anwendung zur Verfügung stellen, werden im Folgendem im Detail analysiert und mittels *UML Use Case Diagram* veranschaulicht.

Ein Anwendungsfall (engl.: use case) beschreibt die Interaktion eines Akteurs mit dem System [19]. Zunächst sollen Akteure, die mit der Anwendung arbeiten, identifiziert werden. Für die zu entwickelnde Anwendung gibt es nur einen Akteur, der nachfolgend **Benutzer** genannt wird.

Die Funktionen der Anwendung lassen sich in drei Gruppen von Anwendungsfällen aufteilen: Benutzer-, Visualisierungs- und Dashboard-Management. Die einzelnen Anwendungsfälle der jeweiligen Gruppe werden im Folgenden beschrieben.

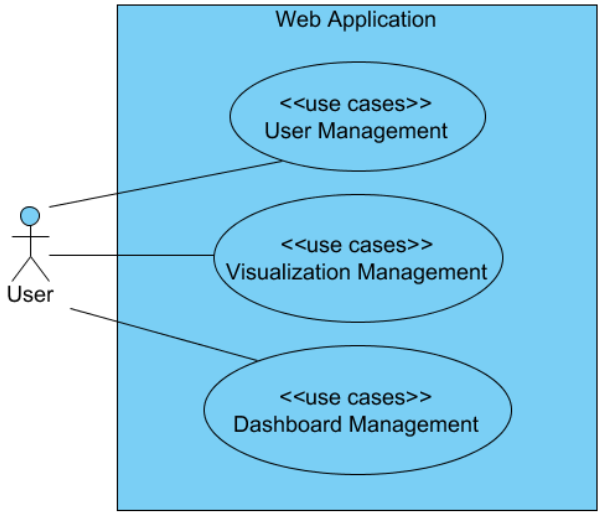


Abbildung 12: Gruppe der Anwendungsfälle

3.3.1 Benutzermanagement

Die Anwendungsfälle in dieser Gruppe ermöglichen einem Benutzer die Registrierung und Anmeldung mit der Software.

3.3.1.1 Anwendungsfall B1: Registrieren

- **Kurzbeschreibung**: Der Benutzer kann sich mit der Anwendung registrieren.
- **Vorbedingungen**: Der Benutzer befindet sich auf dem Registrierungsbereich.
- **Essentielle Schritte**:
    1. Der Benutzer gibt seine Registrierungsdaten an: Name, E-Mail-Adresse, Passwort.
    2. Die Anwendung speichert die Registrierungsdaten des Benutzers.
- **Ausnahmefälle**:
    1. Die E-Mail-Adresse des Benutzers ist bereits vorhanden: Die Anwendung gibt eine Fehlermeldung aus.
- **Nachbedingungen**: Der Benutzer kann sich einloggen.

3.3.1.2 Anwendungsfall B2: Einloggen

- **Kurzbeschreibung**: Der Benutzer kann sich in die Anwendung einloggen.

- **Vorbedingungen**:
    1. Der Benutzer ist registriert.
    2. Der Benutzer befindet sich auf dem Login-Bereich.
- **Essentielle Schritte**:
    1. Der Benutzer gibt seine Daten an: E-Mail-Adresse, Passwort.
    2. Die Anwendung prüft die angegebenen Daten des Benutzers.
    3. Die Anwendung leitet den Benutzer in dessen Benutzerbereich weiter.
- **Ausnahmefälle**:
    1. Die angegebene E-Mail-Adresse oder das Passwort stimmen nicht mit den gespeicherten Daten überein: Die Anwendung gibt eine Fehlermeldung aus.
- **Nachbedingungen**: Der Benutzer befindet sich auf seinem Benutzerbereich.

3.3.1.3 Anwendungsfall B3: Ausloggen

- **Kurzbeschreibung**: Der Benutzer kann sich aus der Anwendung ausloggen.
- **Vorbedingungen**:
    1. Der Benutzer ist eingeloggt.
- **Essentielle Schritte**:
    1. Der Benutzer ruft die Ausloggen-Funktion auf.
    2. Die Anwendung leitet den Benutzer zum Login-Bereich weiter.
- **Ausnahmefälle**: keine
- **Nachbedingungen**: Der Benutzer kann auf seinen Benutzerbereich nicht zugreifen.

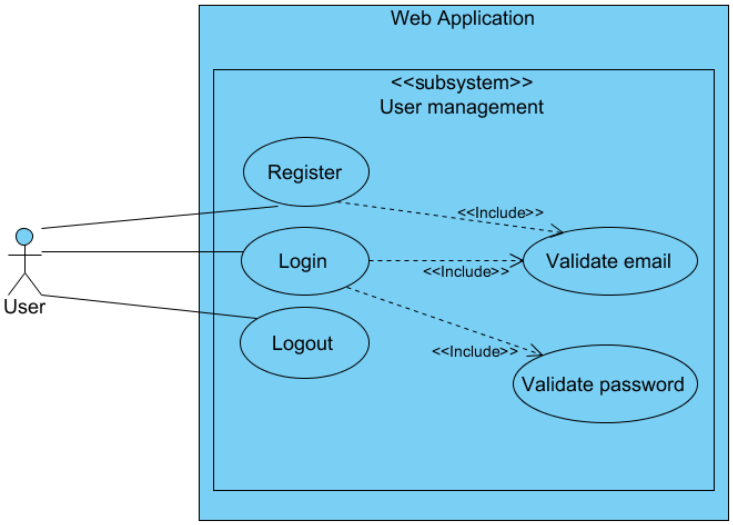


Abbildung 13: Anwendungsfälle Benutzermanagement

3.3.2 Visualisierungsmanagement

Die Anwendungsfälle in dieser Gruppe ermöglichen einem Benutzer die Erstellung, Änderung und Löschung von Visualisierungen. Die dem Benutzer verfügbaren Visualisierungsarten sind Bebauungsplangrafik und Balkendiagram (vgl. 3.2.2 und 3.2.3).

3.3.2.1 Anforderung an die Bebauungsplangrafik

Zur Erstellung einer Bebauungsplangrafik wird folgende Regel definiert und verwendet:

Ein Element Z steht in einer Zelle, die von einem Element X der X-Achse und einem Element Y der Y-Achse definiert ist, genau dann, wenn es eine ***direkte oder indirekte*** *strukturelle Beziehung oder Abhängigkeitsbeziehung (vgl. 2.2.2) zwischen X und Z, Y und Z und X und Y gibt.*

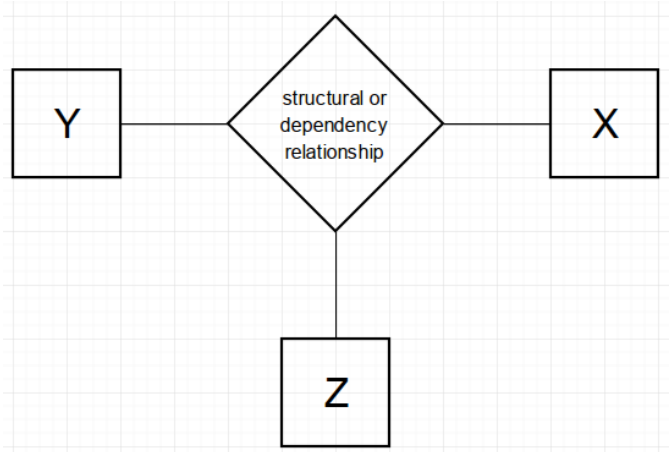


Abbildung 14: Regel zur Erstellung einer Bebauungsplangrafik

Zudem weist Hanschke [10] darauf hin, dass nicht alle Kombinationen von Zeilen, Spalten und Zellen sinnvoll sind. Beispielsweise hilft es nicht, wenn Geschäftsprozesse in den Zellen in Beziehung mit Geschäftsfunktionen und Geräten auf den Achsen gesetzt werden. Eine umfassende Liste von sinnvollen Kombinationen der Elemente in einer Bebauungsplangrafik findet sich in [10], Appendix D. Für die prototypische Implementierung in dieser Arbeit werden Bebauungsplangrafiken mit folgenden Kombinationen bereitgestellt:

Business Function in den Zellen, Business Actor und Business Process in den Zeilen- und Spaltenkopf. Business Actor wird hier als eine Geschäftseinheit verstanden. Diese Kombination stellt einen Zusammenhang innerhalb des Business Layers dar und beantwortet die Fragestellung: Welche Geschäftseinheit nutzt welche fachliche Funktion in welchem Geschäftsprozess?

Application Component in den Zellen, Business Actor und Business Process in den Zeilen- und Spaltenkopf. Diese Kombination stellt einen Zusammenhang zwischen dem Business Layer und dem Application Layer dar und beantwortet die Fragestellung: Welche Geschäftseinheit nutzt welche Anwendung für welchen Geschäftsprozess.

System Software in den Zellen, Business Actor und Business Process in den Zeilen- und Spaltenkopf. Diese Kombination stellt einen Zusammenhang zwischen dem Business Layer und dem Technology Layer dar und beantwortet die Fragestellung: welche Geschäftseinheit wird von welcher Systemsoftware für welchen Geschäftsprozess unterstützt.

System Software in den Zellen, Application Component und Device in den Zeilen- und Spaltenkopf. Diese Kombination stellt einen Zusammenhang zwischen dem Application Layer und dem Technology Layer dar und beantwortet die Fragestellung: welche betriebliche Anwendung wird von welcher Systemsoftware auf welchem Gerät unterstützt.

Folgende Konfigurationsmöglichkeiten stehen dem Benutzer bei der Erstellung einer Bebauungsplangrafik zur Verfügung:

- Auswahl der Elementtypen als Inhalt für die Zellen: *Business Function, Application Component, System Software*.
- Auswahl der entsprechenden Elementtypen für die Zeilen- und Spaltenköpfe: *Business Actor, Business Process, Application Component, Device*.
- Auswahl zur farblichen Hervorhebung des Attributes *health status*.
- Einen Namen für die erstellte Bebauungsplangrafik eingeben.

3.3.2.2 Anforderung an das Balkendiagramm

Zur Demonstration wird das Balkendiagramm verwendet, um die Anzahl von technischen Elementen bezüglich deren Attribut *health status* zu vergleichen. Abbildung 11 zeigt bereits ein Beispiel dafür. Folgende Konfigurationsmöglichkeit steht dem Benutzer bei der Erstellung eines Balkendiagramms zur Verfügung:

- Auswahl eines Elementtyps oder aller Elementtypen: *Application Component, System Software, Device*.
- Einen Namen für das erstellte Balkendiagramm eingeben.

Auf dieser Grundlage werden nachfolgend Anwendungsfälle für das Visualisierungsmanagement beschrieben:

3.3.2.3 Anwendungsfall V1: Visualisierung erstellen

- **Kurzbeschreibung**: Der Benutzer kann eine neue Visualisierung erstellen.

- **Vorbedingungen**: Der Benutzer ist eingeloggt und befindet sich im Visualisierungsmanagementbereich.
- **Essentielle Schritte**:
    1. Der Benutzer gibt an, welche Visualisierungsart er erstellen möchte.
    2. Die Anwendung stellt dem Benutzer die entsprechenden Konfigurationsmöglichkeiten zur Verfügung.
    3. Der Benutzer führt seine gewünschte Konfiguration aus.
    4. Der Benutzer gibt an, dass er die Visualisierung speichern möchte.
    5. Die Anwendung speichert die Visualisierung des Benutzers.
- **Ausnahmefälle**:
    1. Zu der ausgewählten Konfiguration des Benutzers gibt es keine Daten zum Visualisieren: Die Anwendung gibt eine entsprechende Meldung aus.
- **Nachbedingungen**: Der Benutzer kann seine erstellte Visualisierung ansehen.

3.3.2.4 Anwendungsfall V2: Visualisierung bearbeiten

- **Kurzbeschreibung**: Der Benutzer kann eine erstellte Visualisierung ändern
- **Vorbedingungen**: Der Benutzer ist eingeloggt und hat mindestens eine Visualisierung erstellt.
- **Essentielle Schritte**:
    1. Der Benutzer gibt an, dass er eine bestimmte Visualisierung ändern möchte.
    2. Die Anwendung zeigt dem Benutzer die ausgewählte Visualisierung an und stellt die entsprechenden Konfigurationsmöglichkeiten zur Verfügung.
    3. Der Benutzer führt seine gewünschte Konfiguration aus.
    4. Der Benutzer gibt an, dass er die geänderte Visualisierung speichern möchte.
    5. Die Anwendung speichert die geänderte Visualisierung des Benutzers.
- **Ausnahmefälle**:
    1. Zu der ausgewählten Konfiguration des Benutzers gibt es keine Daten zum Visualisieren: Die Anwendung gibt eine entsprechende Meldung aus.
- **Nachbedingungen**: Der Benutzer kann seine geänderte Visualisierung ansehen.

3.3.2.5 Anwendungsfall V3: Visualisierung löschen

- **Kurzbeschreibung**: Der Benutzer kann eine erstellte Visualisierung löschen.
- **Vorbedingungen**: Der Benutzer ist eingeloggt und hat mindestens eine Visualisierung erstellt.
- **Essentielle Schritte**:
    1. Der Benutzer ruft die Lösch-Funktion einer Visualisierung auf.
    2. Die Anwendung löscht die ausgewählte Visualisierung des Benutzers.
- **Ausnahmefälle**: keine
- **Nachbedingungen**: Der Benutzer kann die gelöschte Visualisierung nicht ansehen.

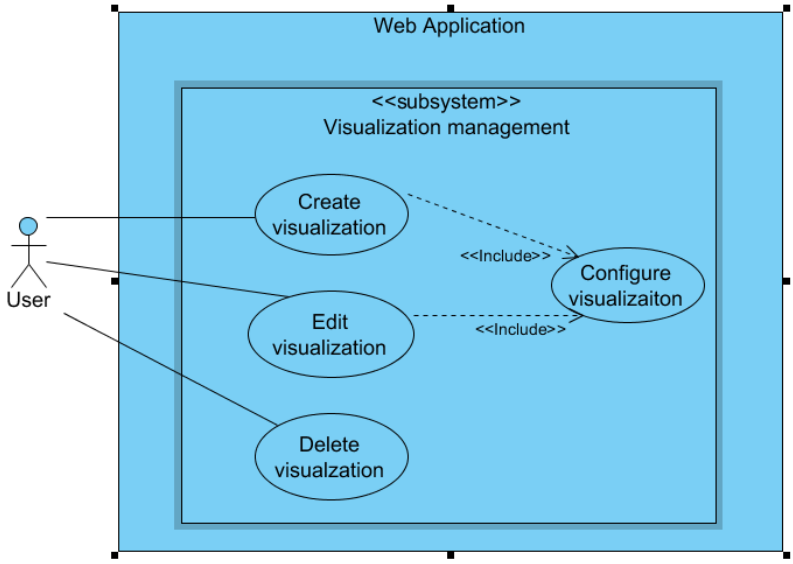


Abbildung 15: Anwendungsfälle Visualisierungsmanagement

3.3.3 Dashboard-Management

Die Anwendungsfälle in dieser Gruppe ermöglichen es dem Benutzer, seine erstellten Visualisierungen einem Dashboard hinzuzufügen. Innerhalb eines Dashboards besitzt ein Benutzer die Möglichkeit, die dazugehörigen Visualisierungen umzupositionieren oder aus dem Dashboard zu entfernen.

3.3.3.1 Anwendungsfall D1: Dashboard erstellen

- **Kurzbeschreibung**: Der Benutzer kann eine neue Visualisierung erstellen.
- **Vorbedingungen**: Der Benutzer ist eingeloggt und befindet sich auf dem Dashboard-Managementbereich.
- **Essentielle Schritte**:
    1. Der Benutzer ruft die Funktion zur Dashboard-Erstellung auf.
    2. Die Anwendung zeigt dem Benutzer seine vorhandenen Visualisierungen an.
    3. Der Benutzer kann folgende Aktionen ausführen:
        - Visualisierungen hinzufügen.
        - Hinzugefügte Visualisierungen umpositionieren.
        - Hinzugefügte Visualisierungen entfernen.
        - Dashboard benennen.
    4. Der Benutzer führt eine oder mehrere Aktionen aus.
    5. Der Benutzer gibt an, dass er das Dashboard speichern möchte.
    6. Die Anwendung speichert das erstellte Dashboard des Benutzers.
- **Ausnahmefälle**: keine

- **Nachbedingungen**: Der Benutzer kann sein erstelltes Dashboard ansehen. Die ausgewählten Visualisierungen sind im Dashboard zu finden.

3.3.3.2 Anwendungsfall D2: Dashboard anzeigen

- **Kurzbeschreibung**: Der Benutzer kann ein erstelltes Dashboard anzeigen lassen.
- **Vorbedingungen**: Der Benutzer ist eingeloggt und hat mindestens ein Dashboard erstellt.
- **Essentielle Schritte**:
    1. Der Benutzer wählt ein Dashboard zur Anzeige aus.
    2. Die Anwendung zeigt dem Benutzer das ausgewählte Dashboard an.
- **Ausnahmefälle**: keine
- **Nachbedingungen**: Der Benutzer kann sein erstelltes Dashboard ansehen. Der Inhalt des Dashboards ist richtig dargestellt.

3.3.3.3 Anwendungsfall D3: Dashboard bearbeiten

- **Kurzbeschreibung**: Der Benutzer kann ein erstelltes Dashboard ändern.
- **Vorbedingungen**: Der Benutzer ist eingeloggt und hat mindestens ein Dashboard erstellt.
- **Essentielle Schritte**:
    1. Der Benutzer wählt ein Dashboard zur Bearbeitung aus.
    2. Die Anwendung zeigt dem Benutzer das ausgewählte Dashboard an.
    3. Der Benutzer kann folgende Aktionen ausführen:
        - Visualisierungen hinzufügen.
        - Hinzugefügte Visualisierungen umpositionieren.
        - Hinzugefügte Visualisierungen entfernen.
        - Dashboard umbenennen.
    4. Der Benutzer führt eine oder mehrere Aktionen aus.
    5. Der Benutzer gibt an, dass er das Dashboard speichern möchte.
    6. Die Anwendung speichert das bearbeitete Dashboard des Benutzers.
- **Ausnahmefälle**: keine
- **Nachbedingungen**: Der Benutzer kann sein bearbeitetes Dashboard ansehen. Änderungen werden sichtbar.

3.3.3.4 Anwendungsfall D4: Dashboard löschen

- **Kurzbeschreibung**: Der Benutzer kann ein erstelltes Dashboard löschen.
- **Vorbedingungen**: Der Benutzer ist eingeloggt und hat mindestens ein Dashboard erstellt.
- **Essentielle Schritte**:

1. Der Benutzer ruft die Lösch-Funktion eines Dashboards auf.
2. Die Anwendung löscht das ausgewählte Dashboard des Benutzers.

- **Ausnahmefälle**: keine
- **Nachbedingungen**: Der Benutzer kann das gelöschte Dashboard nicht ansehen.

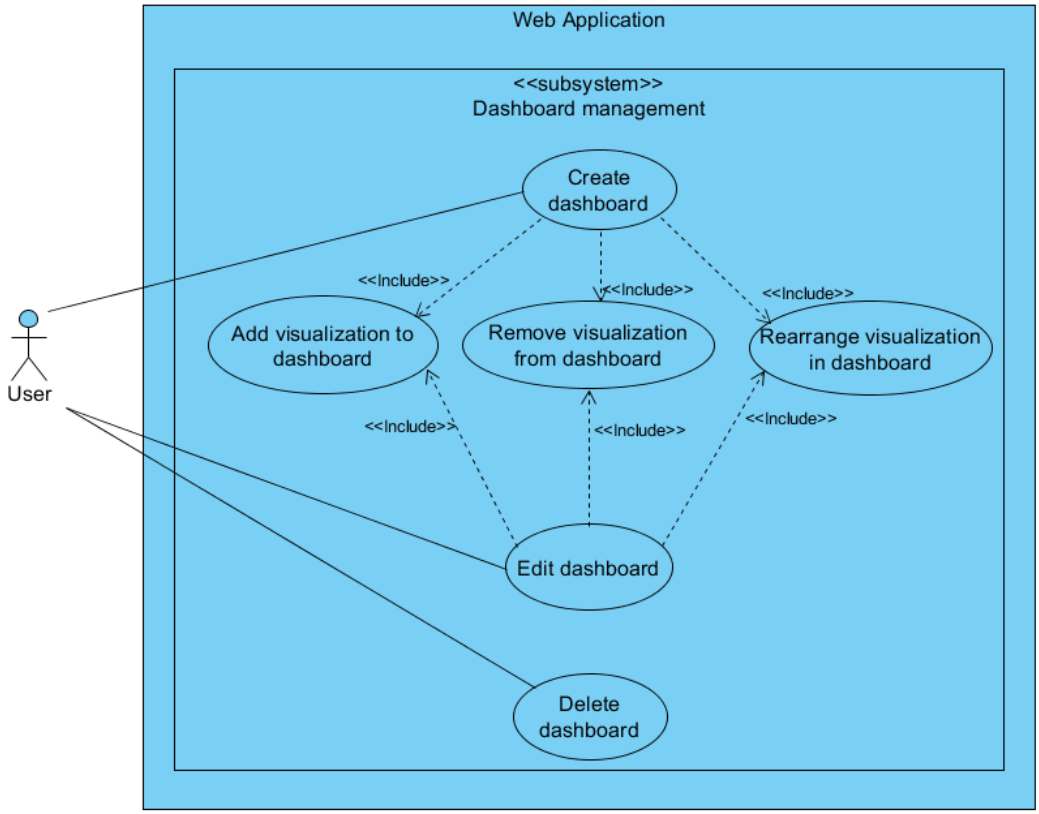


Abbildung 16: Anwendungsfälle Dashboard-Management

3.4 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben Aspekte, die nicht direkt mit den Funktionalitäten einer Software zusammenhängen. Verschiedene Kategorien von nicht-funktionalen Anforderungen lassen sich in der Literatur identifizieren (vgl. [19], [20]), z. B. Effizienz, Benutzerfreundlichkeit, Sicherheit, Performance etc. Da der Fokus in dieser Arbeit jedoch mehr auf den vorgestellten Funktionalitäten liegt, wird im Folgenden nur eine kleine Anzahl von nicht-funktionalen Anforderungen eingeführt, die möglicherweise den Architekturentwurf und die Implementierung der Anwendung beeinflussen können.

Modularität: die Anwendung soll in unabhängige Module aufgeteilt werden, die über entsprechende Schnittstellen interagieren.

Implementierungseinschränkung: die Anwendung soll eine relationale Datenbank verwenden und mit der Programmiersprache Java und JavaScript entwickelt werden.

Sicherheit: Passwörter von Benutzern sollen Dritten nicht im Klartext zugänglich gemacht werden.

Bedienbarkeit: Benutzer sollen auf Mobilendgeräten ohne funktionale Einschränkung mit der Anwendung arbeiten können.

Benutzeroberfläche: die Darstellungssprache der Anwendung ist Englisch.

4 Entwurf

Die zu entwickelnde Anwendung wird basierend auf der Analyse im Kapitel 3 im Nachfolgenden aus verschiedenen Perspektiven genauer betrachtet. Die Benutzeroberfläche wird zuerst mittels Abbildungen veranschaulicht, um die Funktionalität der Anwendung deutlicher zu erklären. Danach wird die Anwendungsarchitektur, bestehend aus Modulen und deren Schnittstellen, vorgestellt. Der algorithmische Ansatz zur Verknüpfung des EA-Metamodells mit den Visualisierungsarten wird währenddessen ebenfalls beschrieben.

4.1 Benutzeroberfläche

Die beschriebenen Funktionen der Anwendung (vgl. 3.3) stehen dem Benutzer in Form einer Webanwendung zur Verfügung. Die Interaktionen vom Benutzer mit der Anwendung werden den vorgestellten Anwendungsfällen entsprechend grafisch erklärt. Es gibt folgende drei Bereiche, in denen sich der Benutzer unterschiedliche Anwendungsfälle zu Nutze machen kann: Benutzermanagement, Visualisierungsmanagement und Dashboard-Management.

4.1.1 Benutzermanagement

Dieser Bereich stellt dem Benutzer die Anwendungsfälle in 3.3.1 bereit.

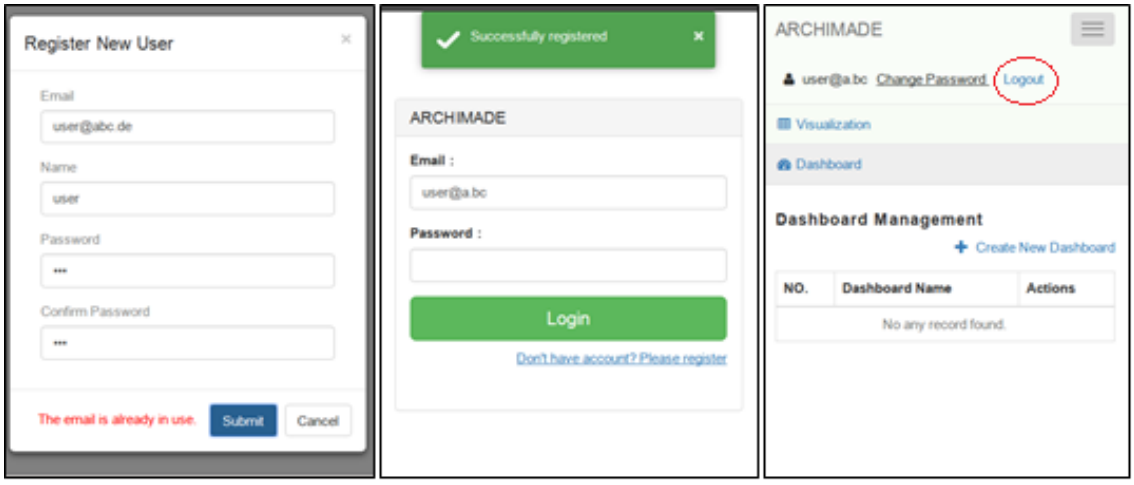


Abbildung 17: Übersicht Benutzermanagement-Bereich

Um sich mit der Anwendung zu registrieren, füllt der Benutzer ein Formular aus, wobei er das Passwort zweimal übereinstimmend eingeben muss. Falls die eingegebene E-Mail-Adresse bereits registriert ist, gibt die Anwendung eine Fehlermeldung aus. Nach der erfolgreichen Registrierung wird der Benutzer zur Einloggen-Seite weitergeleitet. Hier gibt er seine registrierte E-Mail-Adresse und sein Passwort ein, um sich in die Anwendung einzuloggen. Nur beim erfolgreichen Einloggen kann der Benutzer auf seinen eigenen Benutzer-Bereich zugreifen, bestehend aus Menüs zum anderen Bereich und einem Link zum Ausloggen.

4.1.2 Visualisierungsmanagement

Dieser Bereich stellt dem Benutzer die Anwendungsfälle in 3.3.2 bereit.

Um den Visualisierungsmanagement-Bereich aufzurufen, wählt der Benutzer die Menü *Visualization* aus. Hier werden alle erstellten Visualisierungen des Benutzers aufgelistet und der Benutzer hat die Möglichkeit, eine neue Visualisierung zu erstellen, eine vorhandene Visualisierung zu bearbeiten oder sie zu löschen.

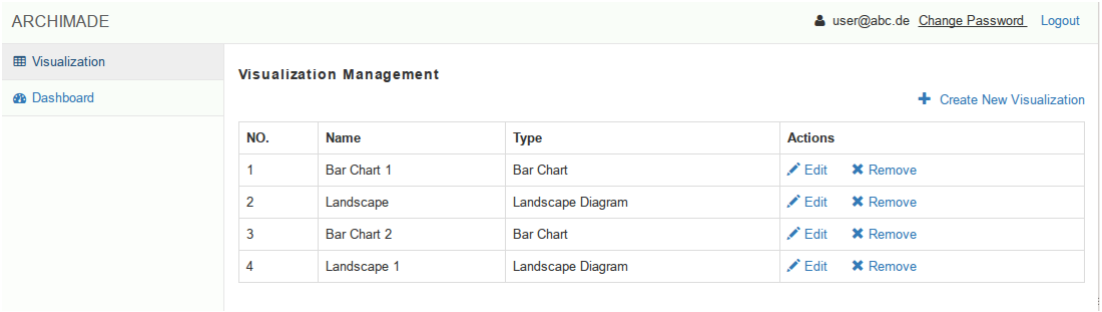


Abbildung 18: Übersicht Visualisierungsmanagement-Bereich

Bei der Erstellung einer neuen Visualisierung werden dem Benutzer unterschiedliche Konfigurationsmöglichkeiten bereitgestellt, je nachdem welche Visualisierungsarten er ausgewählt hat (vgl. 3.2). Abbildung 19 zeigt beispielsweise die Erstellung einer Bebauungsplangrafik, die aus System Software in den Zellen (*Content Type*), Device in den Zeilenköpfen (*Row Type*) und Application Component in den Spaltenköpfen (*Column Type*) besteht. Mit dieser Bebauungsplangrafik kann der Benutzer z. B. einen Handlungsbedarf für das *Message Queueing* System sofort erkennen.

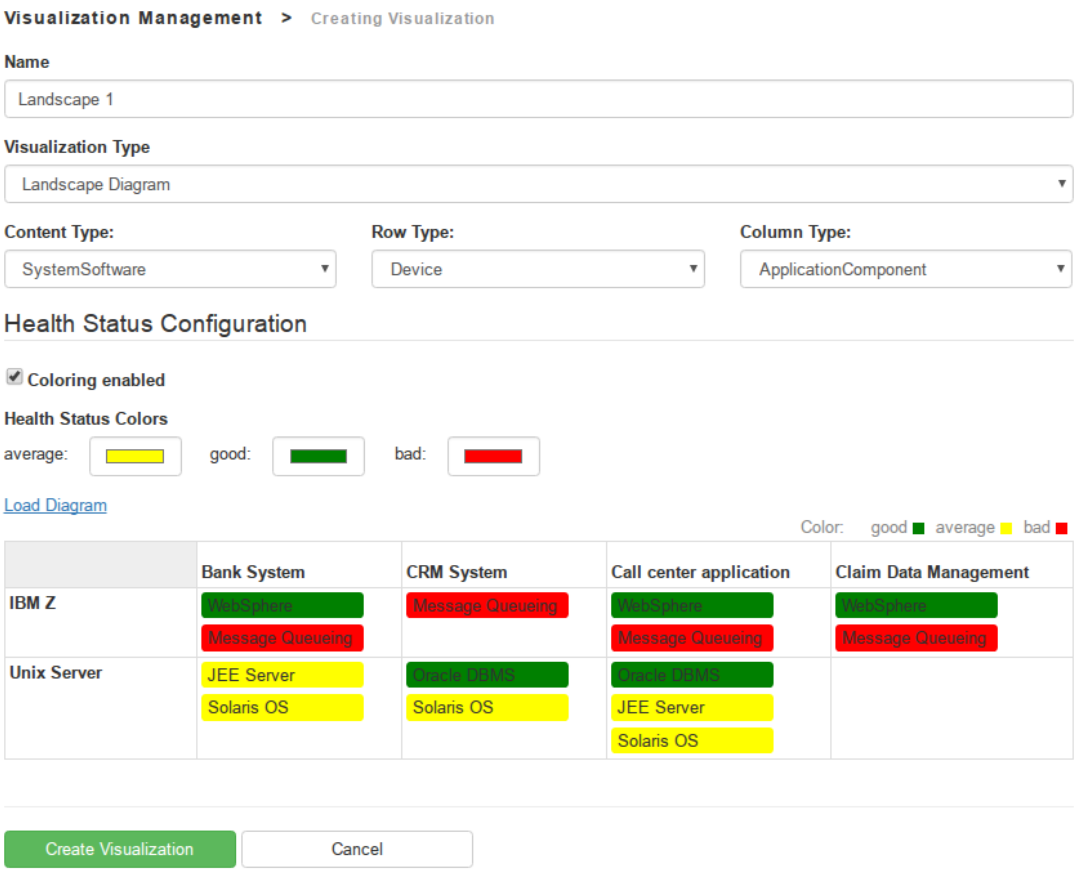


Abbildung 19: Erstellung einer Bebauungsplangrafik

Die Bearbeitung einer vorhandenen Visualisierung wird ähnlich der Erstellung gestaltet.

4.1.3 Dashboard-Management

Dieser Bereich stellt dem Benutzer die Anwendungsfälle in 3.3.3 bereit.

Die erstellten Dashboards des Benutzers werden hier aufgelistet und er kann ein vorhandenes Dashboard öffnen, bearbeiten, löschen oder ein neues erstellen.

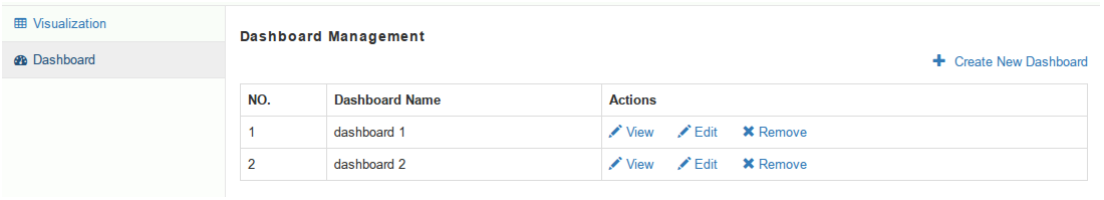


Abbildung 20: Übersicht Dashboard-Management-Bereich

Bei der Erstellung eines neuen Dashboards wählt der Benutzer aus, welche seiner erstellten Visualisierungen er dem Dashboard hinzufügen möchte. Er kann die ausgewählten Visualisierungen wieder aus dem Dashboard entfernen oder mit *drag-and-drop* die Visualisierungen umpositionieren.

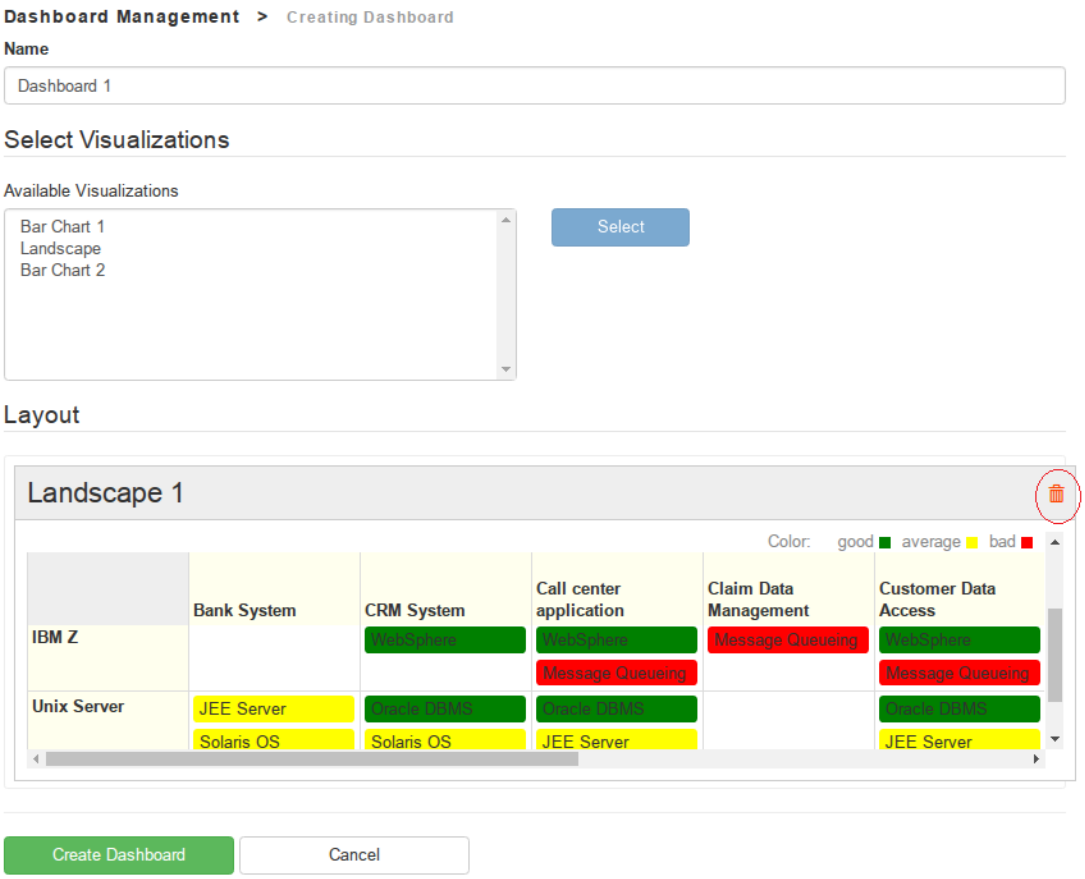


Abbildung 21: Erstellung eines neuen Dashboards

Die Bearbeitung eines erstellten Dashboards erfolgt ähnlich wie die Erstellung. Beim Anzeigen eines erstellten Dashboards kann sich der Benutzer nur das ausgewählte Dashboard ansehen, ohne es zu bearbeiten.

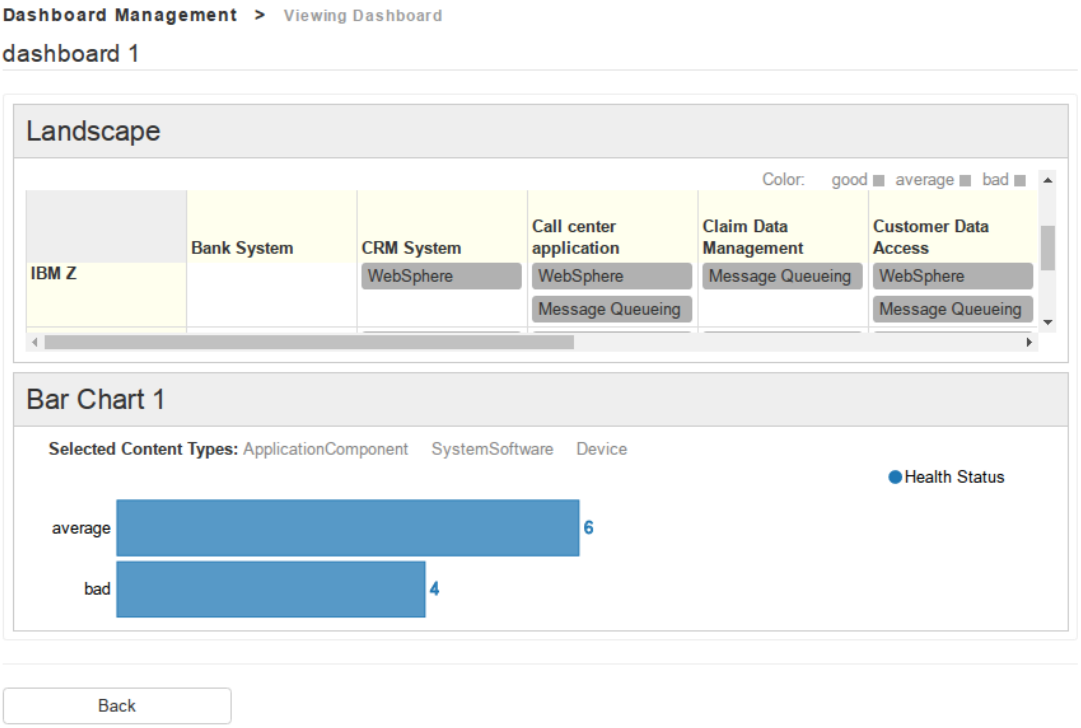


Abbildung 22: Anzeigen eines Dashboards

4.2 Algorithmischer Entwurf

Wie bereits beschrieben (vgl. 3.2.2), können ternäre Beziehungen zwischen Elementen einer Bebauungsplangrafik als direkte Beziehung gespeichert oder als indirekte Beziehung abgeleitet werden. Dieser Abschnitt stellt eine einfache Methode vor, die entsprechend der im Abschnitt 3.3.2.1 definierten Regel indirekte Beziehungen zwischen Architekturelementen ableitet und somit den Inhalt einer beliebigen Bebauungsplangrafik bestimmt.

Gemäß der Spezifikation [8] sind strukturelle Beziehungen und Abhängigkeitsbeziehungen in der ArchiMate-Sprache transitive. Das heißt, wenn strukturelle Beziehungen oder Abhängigkeitsbeziehungen von Element *a* nach *b* und von Element *b* nach *c* bestehen, dann gibt es auch eine strukturelle Beziehung oder Abhängigkeitsbeziehung von Element *a* nach Element *c*. Es wird beachtet, dass Beziehungen in ArchiMate gerichtet sind, weshalb die Formulierung „Beziehung von *Element* nach *Element*“ präziser als „Beziehung zwischen *Element* und *Element*“ ist. Im Folgenden werden den Ausdruck „Beziehung zwischen *a* und *b*“ genutzt, um die Kombination „Beziehung von *a* nach *b* oder von *b* nach *a*“ zu beschreiben.

Wie in der ArchiMate-Spezifikation auch gezeigt wird, ermöglicht die Transitivität eine Kette von Beziehungen mit einer einzigen Beziehung zu ersetzen. Beispielsweise lässt sich in Abbildung 23 eine indirekte Beziehung von *Financial Application* nach *Insourcing and Collections* identifizieren.

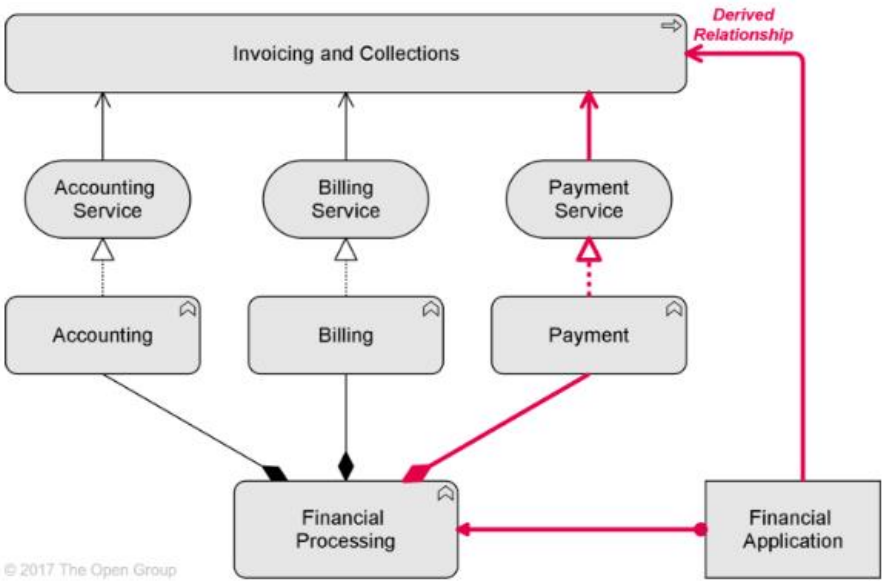


Abbildung 23: Ableitung indirekter Beziehung in ArchiMate [8]

Die Unternehmensarchitektur kann als ein gerichteter Graph betrachtet werden, wobei Elemente als Knoten und Beziehungen als Kanten zu sehen sind. Somit existiert eine Beziehung zwischen Element *a* und Element *b* nur dann, wenn es einen Pfad von *a* nach *b* oder einen Pfad von *b* nach *a* gibt. Verschiedene Algorithmen können verwendet werden, um die Erreichbarkeit zwischen *a* und *b* herauszufinden [21]. Wenn nur die strukturellen Beziehungen und Abhängigkeitsbeziehungen als gerichtete Graph kanten betrachtet werden, dann reicht die Nutzung eine Breitensuche. Eine Funktion ***ist Erreichbar(ab)*** kann beispielsweise eine Breitensuche verwenden, um einen Pfad von ***a*** nach ***b*** zu finden.

Eine Bebauungsplangrafik kann als ein Array von ternären Tupeln in der Form ***x-y-z*** gesehen werden, wobei ***x*** das Element auf der X-Achse, ***y*** das Element auf der Y-Achse und ***z*** das Element in der Zelle, die von ***x*** und ***y*** definiert ist, repräsentiert.

Basierend darauf kann folgender Algorithmus genutzt werden, um eine Bebauungsplangrafik zu konstruieren:

```
für jedes Element x vom Typ X
        für jedes Element y vom Typ Y
                wenn istErreichbar(x,y) oder istErreichbar(y,x), dann
                        für jedes Element z vom Typ Z
                                wenn istErreichbar(x,z) oder istErreichbar(z,x)
                                UND   istErreichbar(y,z) oder istErreichbar(z,y)
                                dann speichert x-y-z in Array ab
```

4.3 Architektur

4.3.1 Architekturübersicht

Da es sich bei dem Prototyp um eine Webanwendung handelt, hat er von Natur aus eine Client-Server-Architektur [19]. Ergänzend dazu erfolgt die Kommunikation zwischen Client und Server in dieser Anwendung mittels des Architekturstiles *REST* (*Repräsentationen State Transfer*) [22]. Im Wesentlichen geht es bei REST um folgende pragmatische Prinzipien für die zu entwickelnde Webanwendung:

Ressourcen werden vom Server dem Client über URLs (Uniform Ressource Locator) zur Verfügung gestellt.

Repräsentationen der Ressourcen werden als Datenformat zum einheitlichen Austausch zwischen Client und Server verwendet. Für diesen Zweck wird in dieser Anwendung das JSON[4]-Format (JavaScript Object Notation) benutzt.

Nachrichten werden zum Austausch von Repräsentationen der Ressourcen zwischen Client und Server verwendet. Für diesen Zweck werden in dieser Anwendung die HTTP-Methoden[5] GET und POST benutzt.

Zustandlos: der Server speichert keinen Zustand des Clients. Nach jeder Request-Response-Runde endet die Kommunikation zwischen Client und Server.

[4] http://www.json.org/. Zugriff am [31.05.2017]
[5] https://tools.ietf.org/rfc/rfc7231.txt Zugriff am [31.05.2017]

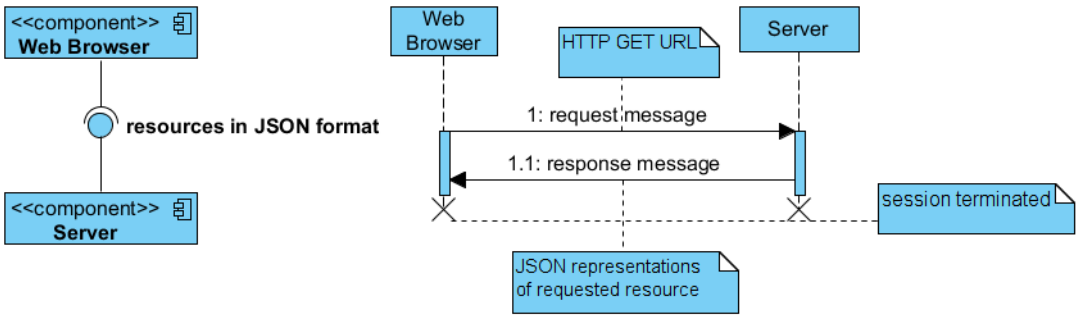


Abbildung 24: Beispiel REST Architekturstil

Der REST-Architekturstil ermöglicht somit eine einfache Kommunikation zwischen Client und Server über vordefinierte URLs und ein einheitliches Datenaustauschformat.

Um die client- sowie serverseitige Datenverarbeitung zu modularisieren, findet das Architekturmuster Model-View-Controller (vgl. [19]) auch Verwendung in diesem Prototyp.

Abbildung 25 stellt die Architekturübersicht der Anwendung dar, die aus drei Hauptkomponenten besteht:

Database: repräsentiert die Datenbank, in der Daten gespeichert werden.

Back-end: repräsentiert den Server, der Ressourcen über URLs bereitstellt.

Front-end: repräsentiert den Client, der Ressourcen vom Server abfragt. Der Client stellt dem Benutzer die Funktionen der Anwendung über Web-Seiten bereit.

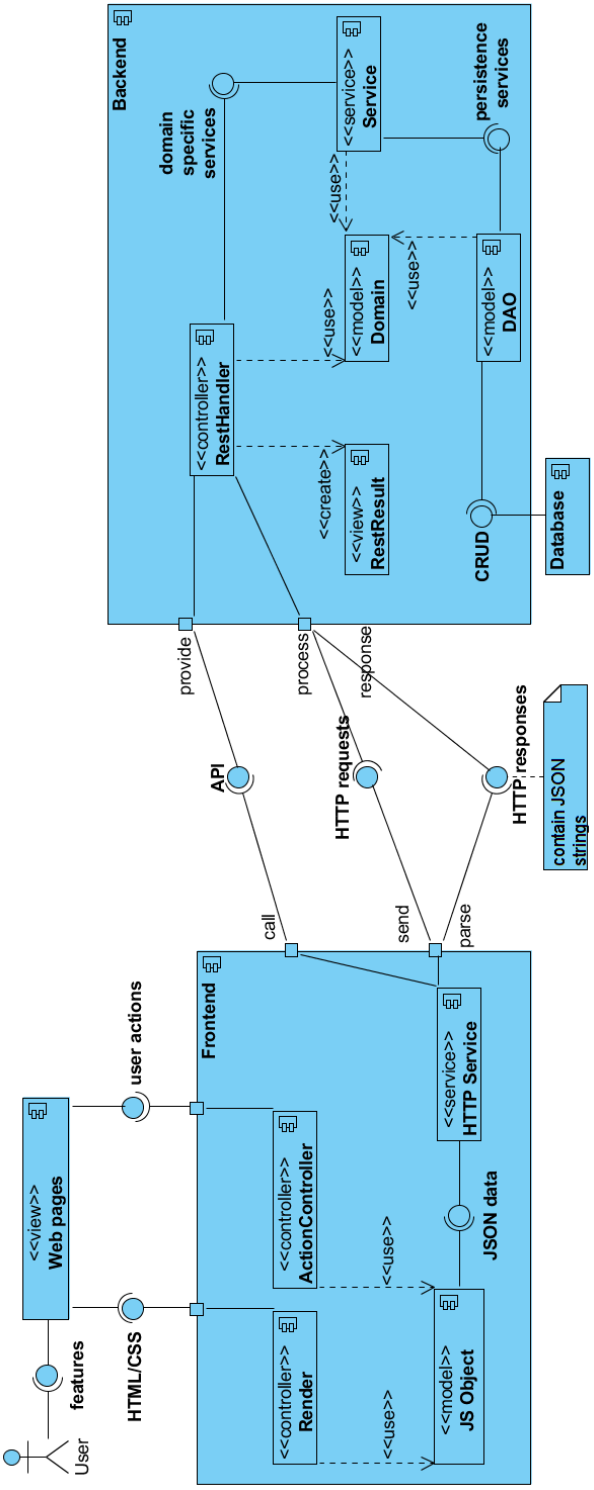


Abbildung 25: Übersicht der Architektur

Auf die Schnittstellen und die interne Struktur der jeweiligen Hauptkomponenten wird im Folgenden eingegangen.

4.3.2 Schnittstellenbeschreibung

Das Back-end stellt verschiedene Dienste zur Verfügung, die das Front-end über vordefinierte URLs abfragen kann. Antworten vom Back-end werden an das Front-end im JSON-Format mit folgender Struktur gesendet:

{ "code" : "<Verarbeitungsstatus>" , "data" : <Ressourcen im JSON-Format> }

Der HTTP-Status-Code 200 wird immer an das Front-end gesendet, um zu zeigen, dass das Back-end gestartet und richtig konfiguriert wurde. Der anwendungsspezifische Verarbeitungsstatus wird über das Attribut *code* repräsentiert und hat folgende mögliche Werte:

ok - Abfrage vom Front-end wurde erfolgreich bearbeitet.
internal_error - Das Back-end hat eine interne Exception geworfen.
invalid_user - Die E-Mail oder das Passwort vom Benutzer ist nicht falsch eingegeben.
existing_user - Die eingegebene E-Mail ist bereits in der Datenbank gespeichert.

Der HTTP-Status-Code 500 wird an das Front-end gesendet, wenn das Back-end nicht gestartet oder nicht richtig konfiguriert wurde.

Das Attribut *data* stellt Ressourcen dar, die das Front-end abgefragt hat.

Die verfügbare URLs, die das Back-end bereitstellt, werden nachfolgend detailliert beschrieben.

4.3.2.1 Element

Über die URLs dieser Schnittstelle können EA-Daten gelesen werden.

Alle Elemente zurückliefern	
Methode	**GET**
URL	/element/all.do
Parameter	Keine
Beispiel Response	"data":{ "1c1b0c6":{

	element_id: "1c1b0c6",
	element_name: "SQL Server"
	type: {
	element_type_id: 2,
	element_type: "SystemSoftware",
	layer_id: null
	},
	},
	…
	}

Alle Elemente von bestimmten Typen zurückliefern	
Methode	**GET**
URL	/element/listByTypes.do?types=<type1>,<type2>,…
Parameter	types: Typen der zurückzuliefernden Elemente
Beispiel	/element/listByTypes.do?types=BusinessActor,BusinessProcess

Beispiel Response	"data":{
	"ApplicationComponent":[
	{ "element_id":"id-4974",
	"element_name":"Financial Application",
	"type":{
	"element_type_id":18,
	"element_type":"ApplicationComponent",
	"layer_id":null
	}
	},
	…],
	"SystemSoftware":[…],
	…
	}

Alle Elemente von bestimmten Typen zählen und nach einem bestimmten Attribut gruppieren	
Methode	**GET**

URL	/element/groupByProperty.do?propId=<propertyId>&types=<type1>,<type2 ,…
Parameter	propId: ID des gesuchten Attributs types: Typen der zu gruppierenden Elemente
Beispiel	/element/groupByProperty.do?propId=1001&types=ApplicationComponent' 2CSystemSoftware
Beispiel Response	"data":[{"total":5,"propertyValue":"average"}, {"total":3,"propertyValue":"bad"}, {"total":9,"propertyValue":"good"}, {"total":2,"propertyValue":"unspecified"}]

Alle Elemente und deren Attribut-Werte zurückliefern	
Methode	**GET**
URL	/element/withProperty.do
Parameter	Keine
Beispiel	/element/withProperty.do

Beispiel Response	data":{
	"id-4994+2001":{
	"element_id":"id-4994","value":"good","property_id":1001
	},
	…
	}

Alle Elementtypen zurückliefern	
Methode	**GET**
URL	/element/types.do
Parameter	Keine
Beispiel	/element/types.do

Beispiel Response	"data":{
	"CommunicationNetwork":{
	"element_type_id":20,
	"element_type":"CommunicationNetwork",
	"layer_id":5
	},
	"BusinessEvent":{
	"element_type_id":14,
	"element_type":"BusinessEvent",
	"layer_id":2
	},
	…
	}

Alle Elemente einer Bebauungsplangrafik zurückliefern	
Methode	**GET**
URL	/element/landscape.do?rowType=<row>&colType=<col>&cellType=<cell>
Parameter	rowType: Typ der Elemente im Zeilenkopf colType: Typ der Elemente im Spaltenkopf cellType: Typ der Elemente in Zellen
Beispiel	/element/landscape.do?rowType=ApplicationComponent&colType=Device& ellType=SystemSoftware
Response	Ein Array von ternären Tupeln in der Form <Zeile element_id>,<Spalte element_id>,<Zelle element_id>
Beispiel Response	"data": ["id-4974,03ec916,fe5de85", "id-4974,03ec916,id-5000", …]

4.3.2.2 Benutzer

Einloggen	
Methode	**POST**

URL	/public/user/login.do
Parameter	email: E-Mail-Adresse des Benutzers password: Passwort des Benutzers
Beispiel	POST /public/user/login.do Content-Type:application/json;charset=UTF-8 {"email":"user@abc.de","password":"123"}
Response	{"code":"ok"} Erfolgreich eingeloggt {"code":"invalid_user"} E-Mail oder Password ist falsch eingegeben

Registrieren	
Methode	**POST**
URL	/public/user/register.do
Parameter	email: E-Mail-Adresse des Benutzers name: Name des Benutzers password: Passwort des Benutzers
Beispiel	POST /public/user/register.do Content-Type:application/json;charset=UTF-8 {"email":"user@abc.de" , "name":"user" , "password":"123"}
Response	{"code":"ok"} Erfolgreich registriert {"code":"existing_user"} E-Mail ist bereits vorhanden

4.3.2.3 Visualisierung

Jede Visualisierung verfügt über Attribute, die beim Aufruf der URLs eventuell angegeben werden müssen.

id: eine eindeutige Kennzeichnung der Visualisierung, die bei der Erstellung automatisch generiert wird.
name: Der vom Benutzer eingegebene Name einer Visualisierung.
type: Visualisierungsart. Erlaubte Werte sind *Landscape Diagram* (Bebauungsplangrafik) und *Bar Chart* (Balkendiagramm).
config: Konfiguration und Inhalt einer Visualisierung. Für jede Visualisierungsart werden weitere individuelle Sub-Attribute von *config* benötigt.

Bebauungsplangrafik-Attribute:

contentType: Typ der Elemente in Zellen.
rowType: Typ der Elemente im Zeilenkopf.
colType: Typ der Elemente im Spaltenkopf.
coloring: boolescher Wert, der angibt, ob farbliche Hervorhebung erfolgen soll.
colorMap: Legende der farblichen Hervorhebung.
diagramData: ein Array von ternären Tupeln in der Form
<Zeile element_id>,<Spalte element_id>,<Zelle element_id>

```
{
   "name":"landscape",
   "type":"Landscape Diagram",
   "config":"{
       "contentType":"BusinessFunction",
       "rowType":"BusinessActor",
       "colType":"BusinessProcess",
       "coloring":true,
       "colorMap": {
           "good":"#008000","average":"#ffff00","bad":"#ff0000"
       },
       "diagramData":[
           "f6083a4,id-4955,id-4933",
           "f6083a4,id-4960,id-4933",
           …
       ]
    }"
}
```

Abbildung 26: Attribute einer Bebauungsplangrafik - Beispiel

Balkendiagramm-Attribute:

propId: ID des zugehörigen Attributes, in diesem Prototyp ist es nur *health status*.
contentTypes: Attribut-Werte von Elementen welcher Typen werden visualisiert.

```
{
  "name":"bar chart",
  "type":"Bar Chart",
  "config":"{
     "propId":1001,
     "contentTypes":[
          "ApplicationComponent",
          "SystemSoftware"
     ]
  }"
}
```

Abbildung 27: Attribute eines Balkendiagramms - Beispiel

Visualisierung erstellen	
Methode	**POST**
URL	/visualize/create.do
Parameter	*name*, *type* und *config* wie oben beschrieben.
Beispiel	vgl. Abbildung 25 und Abbildung 26
Response	{"code":"ok"} Visualisierung wurde erfolgreich erstellt

Visualisierung bearbeiten	
Methode	**POST**
URL	/visualize/update.do
Parameter	*name*, *type* und *config* wie oben beschrieben *id*: ID der zu bearbeitende Visualisierung

Beispiel	POST /visualize/update.do
	Content-Type:application/json;charset=UTF-8
	{
	"name":"barChart",
	"type":"Bar Chart",
	"config":"{
	"propId":1001,"contentTypes":["ApplicationComponent"]}",
	"id":21

	}
Response	{"code":"ok"} Visualisierung wurde erfolgreich bearbeitet

Visualisierung löschen	
Methode	**GET**
URL	/visualize/delete.do?id=<id>
Parameter	*id*: ID der zu löschende Visualisierung
Beispiel	/visualize/delete.do?id=21
Response	{"code":"ok"} Visualisierung ist erfolgreich gelöscht

4.3.2.4 Dashboard

Ein Dashboard besteht aus mehreren Visualisierungen. Ähnlich einer Visualisierung verfügt ein Dashboard auch über die Attribute *id, name* und *config*, wobei das Attribut *config* ein Array ist, in dem jedes Array-Element beschreibt, welche Visualisierung wo und wie in einem Dashboard geladen wird, z. B.

```
"config":"
[
    {"name":"bar1","sizeX":20,"sizeY":7,"vId":"18","row":0,"col":0},
    …
]"
```

Abbildung 28: *config* Attribute eines Dashboards - Beispiel

name: Name der Visualisierung, die dem Dashboard zugeordnet ist.
sizeX: Breite der Visualisierung.
sizeY: Höhe der Visualisierung.
vId: ID der Visualisierung.
row und *col*: relative Position der Visualisierung in dem Dashboard.

Dashboard erstellen	
Methode	**POST**

URL	/dashboard/create.do
Parameter	*name* und *config* wie oben beschrieben.
Beispiel	vgl. Abbildung 27
Response	{"code":"ok"} Dashboard wurde erfolgreich erstellt

Dashboard bearbeiten	
Methode	**POST**
URL	/dashboard/update.do
Parameter	*name*, *type* und *config* wie oben beschrieben *id*: ID des zu bearbeitenden Dashboards

Beispiel	POST /dashboard/update.do
	Content-Type:application/json;charset=UTF-8
	{
	"name":"d3",
	"config":"[
	{"name":"b1","sizeX":20,"sizeY":7,"vId":"18","row":0,"col":0}
	]",
	"id":5
	}

Response	{"code":"ok"} Dashboard wurde erfolgreich bearbeitet

Dashboard löschen	
Methode	**GET**
URL	/dashboard/delete.do?id=<id>
Parameter	*id*: ID des zu löschenden Dashboards
Beispiel	/dashboard/delete.do?id=5
Response	{"code":"ok"} Dashboard ist erfolgreich gelöscht

4.3.3 Datenmodell

Die Daten der Unternehmensarchitektur werden auf einer relationalen Datenbank gespeichert. Das konkrete Datenbankschema wird in Abbildung 29 dargestellt. Die Tabellen sind dem vorgestellten EA-Metamodell (vgl. 3.1) entsprechend aufgebaut.

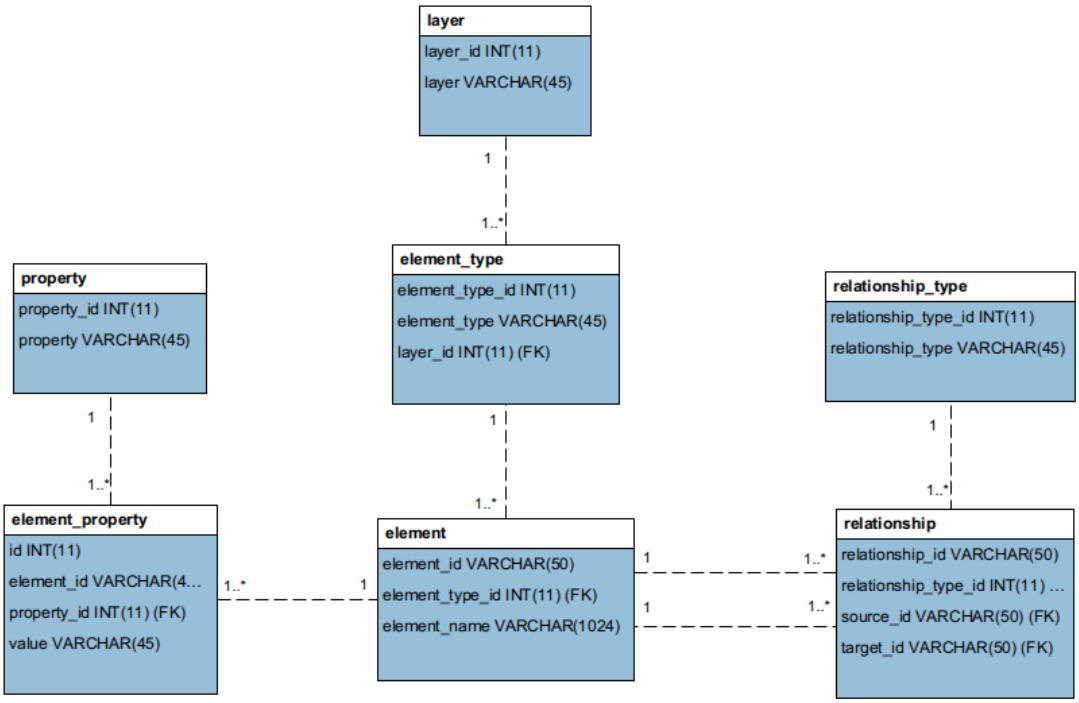


Abbildung 29: Das Datenmodell der Unternehmensarchitektur

Zu beachten gilt, dass in der Tabelle *relationships* nur direkte Beziehungen zwischen Elementen gespeichert werden. Auf Basis der direkten Beziehungen und vordefinierter Regeln können beliebige indirekte Beziehungen abgeleitet werden. Ergänzend zu dem EA-Metamodell wird die Tabelle *element_property* verwendet, um Attribut-Werte von Elementen zu speichern, sofern sie vorhanden sind.

Für die Speicherung der Anwendungsdaten wird das folgende Datenbankschema definiert und verwendet:

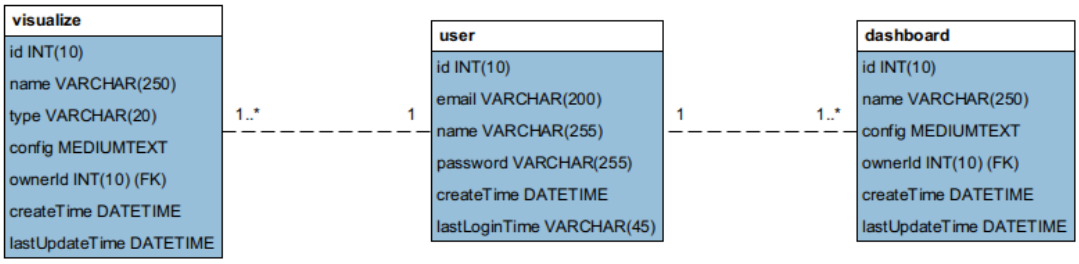


Abbildung 30: Das Datenmodell der Anwendung

Die Tabelle user speichert Login-Daten von Benutzern, wobei Passwörter hier verschlüsselt werden.

Die Tabelle visualize speichert Daten über erstellte Visualisierungen eines Benutzers. Mit dem Attribut config werden Konfigurationsdaten und Inhalt einer Visualisierung direkt im JSON-Format gespeichert.

Die Tabelle dashboard speichert Daten über erstellte Dashboards eines Benutzers. Ein Dashboard kann keine oder mehrere Visualisierungen enthalten. Eine Visualisierung kann keinem oder mehreren Dashboards(s) zugewiesen werden. Diese m:n-Beziehung wird nicht in dem Datenbankschema modelliert, sondern direkt über das Attribut config abgespeichert. Somit wird Datenbankabfragen vereinfacht, was aber möglicherweise zu komplexerer Anwendungslogik führen könnte.

4.3.4 Back-end

Das Back-end verarbeitet Schnittstellen-Aufrufe und stellt Ressourcen, die auf der Datenbank gespeichert werden, dem Front-end zur Verfügung. Der Aufbau des Back-ends besteht aus verschiedenen Komponenten und wird im Folgenden beschrieben.

4.3.4.1 RestResult

Diese Komponente repräsentiert das Ergebnis, das in der HTTP-Response an das Front-end gesendet wird. RestResult stellt nur die Attribute *code* und *data* dar und verfügt über keine Funktionen.

4.3.4.2 Domain Objects

Entitäten des Datenmodells werden in dieser Komponente abgebildet. Die Domain Objects verfügen über ähnliche Attribute wie im Datenmodell beschrieben und Zugriffsfunktionen, die diese Attribute abfragt oder ändert.

4.3.4.3 Data Access Objects (DAO)

Die *Data Access Objects* kommunizieren mit der Datenbank und stellt anderen Komponenten Funktionen, die zur Abfrage oder Speicherung von Domain Objects auf der Datenbank genutzt werden können, zur Verfügung. Die DAO-Komponente benutzt Zugriffsfunktionen der Domain Objects, um sie mit den Datenbanktabellen abzubilden.

4.3.4.4 Service

Die Anwendungslogik wird in dieser Komponente realisiert, z. B. die Ableitung von indirekten Beziehungen zwischen Architekturelementen. Service kann Domain Objects erstellen oder manipulieren und benutzt Funktionen von DAO, um Daten aus der Datenbank abzufragen oder zu speichern.

4.3.4.5 RestHandler

Diese Komponente ist zuständig für die Verarbeitung von Abfragen des Front-ends und macht sich die Service zunutze, um Ergebnisse für das Front-end zu kalkulieren. RestHandler aggregiert verschiedene Domain Objects, um RestResult zu erstellen und an das Front-end zu schicken. Für jede beschriebene Schnittstellengruppe (vgl. 4.2.2) gibt es einen entsprechenden RestHandler, z.B. BenutzerRestHandler oder DashboardRestHandler.

4.3.5 Front-end

Das Front-end ruft zum einen URLs des Back-ends auf, um Ressourcen abzufragen oder zu ändern. Zum anderen ist das Front-end zuständig für die Interaktion mit dem Benutzer. Folgende Sub-Komponenten sind Bestandteil des Front-ends:

4.3.5.1 HTTP Service

Diese Komponente übernimmt die HTTP-Kommunikation zwischen Front-end und Back-end. Schnittstellenaufrufe werden als vom HTTP Service als HTTP-Request an das Back-end gesendet. Enthaltene JSON-Daten in der HTTP-Response vom Back-end stellt anderen Komponenten zur Verfügung.

4.3.5.2 JavaScript Objects

Zurückgelieferte Ressourcen vom Back-end werden über JavaScript Objects abgebildet. Die JavaScript Objects verfügen auch über Zugriffsfunktionen und lassen sich damit von anderen Komponenten manipulieren oder abfragen.

4.3.5.3 Render

Render erstellt JavaScript Objects und verwendet deren Funktionen, um entsprechende Web-Seiten darzustellen. Für jede Seite und Visualisierungsart gibt es einen entsprechenden Render.

4.3.5.4 ActionController

Der ActionController nimmt die Interaktionen des Benutzers (z. B. Klicken, Scrollen) entgegen und ändert die JavaScript Objects dementsprechend, um die Interaktionen als Parameter an den HTTP Service weiterzugeben.

5 Implementierung

In diesem Kapitel werden die relevanten Aspekte für die Implementierung des Prototyps gezeigt. Verwendete Technologie und Software Frameworks werden zunächst kurz vorgestellt. Auf Basis eines Szenarios lassen sich die Interaktionen zwischen den im Kapitel 4 beschriebenen Komponenten anhand einem UML Sequenz-Diagramm und Code-Beispiele darstellen.

5.1 Verwendete Frameworks

Der Prototyp wurden in zwei Teilprojekten entwickelt, wobei das Back-end Java und das Front-end JavaScript verwendet. Die Entwicklung macht sich verschiedene Frameworks zunutze. Ein Software Framework bietet vordefinierte Struktur an und stellt verschiedene, wiederverwendbare Komponenten zur Verfügung, damit die Entwicklung neuer Anwendungen erleichtert werden.

Die Komponente Back-end wurde mithilfe von dem Framework Spring[6] entwickelt. Das Modul Spring MVC übernimmt zum einen die Verteilung von HTTP-Request an die anwendungsspezifischen Controller, die mit entsprechenden Annotationen gekennzeichnet sind. Zum anderen werden HTTP-Response auch vom Spring MVC automatisch an das Front-end geschickt. Der Benutzer des Frameworks kann sich somit auf die Implementierung der Anwendungslogik konzentrieren. Auch zur Speicherung und Verarbeitung von Anwendungsdaten bietet Spring viele Möglichkeiten an. Spring zusammen mit dem Framework MyBatis[7] ermöglicht eine unkomplizierte Abbildung zwischen Domain Objects und relationalen Datenbanktabellen. Hierfür muss die Komponente DAO lediglich angeben, welche Domain Objects mit welchen Tabellen verknüpft werden sollen. Die Verbindung mit der Datenbank und die Ausführung von SQL-Abfragen übernimmt das Framework.

Das Front-end macht sich das JavaScript Framework AngularJS[8] zunutze. AngularJS verfolgt den Ansatz vom Model-View-Controller-Muster und ermöglicht somit eine klare Trennung zwischen Anwendungslogik mit JavaScript und Datenrepräsentation mit HTML.

---

[6] https://spring.io/ Zugriff am [31.05.2017]
[7] http://www.mybatis.org/mybatis-3/ Zugriff am [31.05.2017]
[8] https://angularjs.org/ Zugriff am [31.05.2017]

5.2 Implementierung eines Szenarios

Die Kommunikation und Interaktionen zwischen den Anwendungskomponenten werden mittels des Szenarios *Bebauungsplangrafik laden* veranschaulicht. In diesem Szenario gibt der Benutzer an, aus welchen Elementtypen die Bebauungsplangrafik besteht, dann zeigt ihm die Anwendung dementsprechend den Inhalt der Bebauungsplangrafik an.

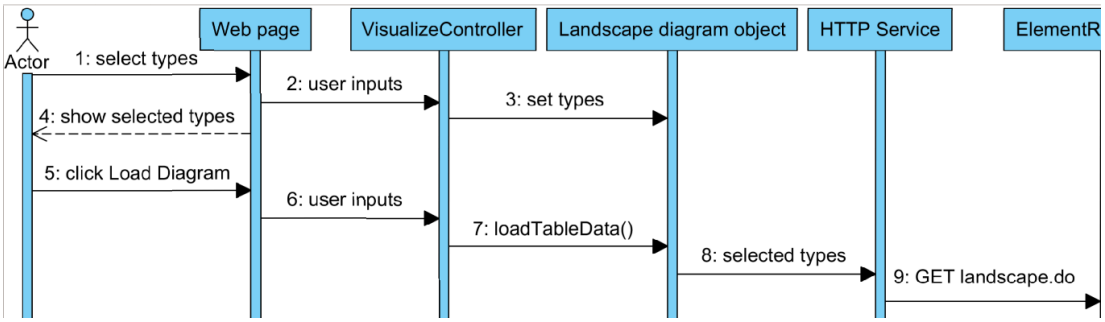


Abbildung 31: Bebauungsplangrafik laden – Teil 1

Zuerst werden dem Benutzer die HTML-Form zur Erstellung einer Bebauungsplangrafik gezeigt, mit der er den Inhalttyp, Spaltentyp und Zeilentyp auswählen kann. Die Auswahl des Benutzers wird von der Komponente ***VisualizeController*** als Attribute eines entsprechenden JavaScript Object über das sogenannte Directive ***ng-model*** gemappt.

```
<div ng-show="type == 'Landscape Diagram'">
        <div class="form-group">
            <label>Content Type:</label>
            <select class="form-control" ng-model="diagram.form.contentType">
              <option value="">Please select content type</option>
              <option ng-repeat="type in diagram.form.getEnabledContentTypes()"
                   value="{{type}}">{{type}}
              </option>
            </select>
          </div>
        <a href="javascript:void(0)" ng-click="diagram.form.loadTableData()">
                Load Diagram
        </a>
```

Nachdem der Benutzer die Typen für seine Bebauungplangrafik ausgewählt hatte, klickt er auf den Link *Load Diagram*, der auch von *VisualizeController* mittels das Angular Directive ***ng-click*** mit der Methode ***loadTableData()*** des JavaScript Object namens ***diagram*** verknüpft wird. Die Komponente *VisualizeController* befindet sich im Ordner */src/app/controllers/visualize* des Front-end Projekts.

```
class VisualizeController {
  constructor(baseAjax, $scope, storage, $uibModal, toastr) {
        // ...
    $scope.diagram = {
        loadTableData: ()=> {
          baseAjax.get('/element/landscape.do?rowType=' + form.rowType) +
              '&colType=' +form.colType) + '&cellType=' + form.contentType))
            .then(res=> {
              $scope.diagram.form.diagramData = res.data;
            })
        },
```

In der Methode ***loadTableData()*** werden die Benutzer-Inputs als Parameter für den URL-Aufruf an die Komponente HTTP Service übergeben. Der HTTP Service namens ***baseAjax***, der im Ordner */src/app/services* sich befindet, schickt dann ein HTTP-Request **GET** an das Back-end.

Das HTTP-Request wird danach vom Back-end verarbeitet.

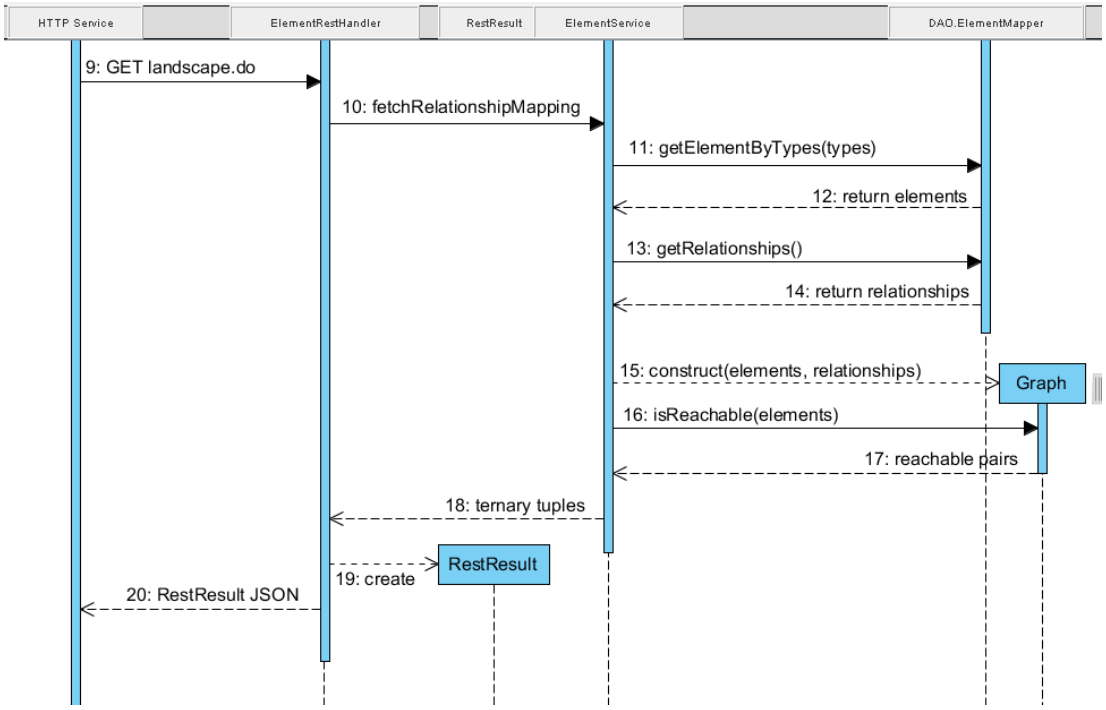


Abbildung 32: Bebauungsplangrafik laden - Teil 2

Die Komponente *ElementRestHanlder* ist mit der Annotation *@RestController* sowie *@RequestMapping(landscape.do)* gekennzeichnet und bekommt deswegen von dem

Framework Spring MVC das HTTP-Request zugewiesen. Die HTTP-Request Parameter werden auch mittels Annotation gekennzeichnet und an die Java-Variables gebunden.

```
@RestController
@RequestMapping("/element")
public class ElementRestHandler extends BaseAction {
        @RequestMapping("/landscape.do")
        public RestResult diagram(HttpServletRequest request,
                        @RequestParam(value = "rowType") String rowType,
                        @RequestParam(value = "colType") String colType,
                        @RequestParam(value = "cellType") String cellType) {
                try {
                        List<String> list = elementService.fetchRelationshipMapping(
                                        rowType, colType, cellType);
                        return new RestResult("ok", list);
                } catch (Exception e) {
                        logger.error("", e);
                        return new RestResult("internal_error");
                }
        }
```

ElementRestHanlder verwendet die Methode *fetchRelationMapping()* der *ElementService* Klasse um den Inhalt der gewünschten Bebauungsplangrafik des Benutzers zu liefern. *ElementService* nutzt wiederum das Data Acces Object *ElementMapper*, um Daten aus der Datenbank abzufragen. Alle Elemente der ausgewählten Typen und deren strukturellen Beziehungen sowie Abhängigkeitsbeziehungen stellt das DAO *ElementMapper* zur Vergügung. Die Klasse *ElementService* erstellt daraus einen gerichteten Graph und verwendet den vorgestellten Algorithmus im Abschnitt 4.2, um eine Liste von ternären Tupeln, die den ganzen Inhalt der Bebauungsplangrafik repäsentiert, zu generieren.

```
@Service
public class ElementService {
        public List<String> fetchRelationshipMapping(rowType, colType, contentType) {
                elementMapper.getElementByTypes(rowType);
                elementMapper.getElementByTypes(colType);
                elementMapper.getElementByTypes(contentType);
                elementMapper.getRelatoinships();
                …
                Graph g = new Graph(idSet.size());
                for (Relationship relationship : rList) {
                        g.addEdge(…);
                }
                // Der Algorithmus im Abschnitt 4.2
```

Die ternären Tupeln bekommt der *ElementRestHanlder* zurückgeliefert und daraus wird eine Instanz der Komponente *RestResult* erzeugt. Das Spring Framework wandelt diese *RestResult* Instanz automatisch ins JSON-Format um und sendet einen HTTP-Response zurück an das Front-end.

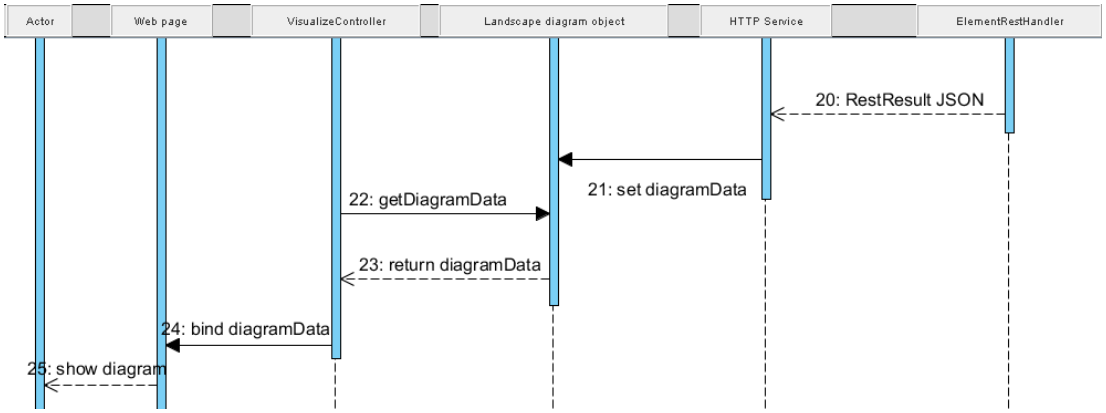


Abbildung 33: Bebauungsplangrafik laden - Teil 3

Den HTTP-Response nimmt der *HTTP-Service* entgegen und extrahiert Daten daraus, um die entsprechenden Attribute des JavaScript Object *diagram* zu ändern. Der *VisualizeController* bindet das Object wieder mit dem HTML-Code und dem Benutzer wird eine Bebauungsplangrafik mit den geladenen Daten angezeigt.

```
class VisualizeController {
      $scope.diagram = {
                  $scope.diagram.form.diagramData = restResult.data;
               })
          }
...
// HTML-Code
<tr ng-repeat="rowEle in diagram.form.getRows()">
                  <td>{{rowEle.element_name}}</td>
                  <td ng-repeat="colEle in diagram.form.getCols()">
                    <div class="content-item" ng-repeat="contentId in
                          diagram.form.getContents(rowEle.element_id,colEle.element_id)">
                       {{allElements[contentId].element_name}}
                     </span>
```

Die Implementierung von anderen Komponenten und Anwendungsfällen erfolgt ähnlich, wobei viele Hintergrund-Arbeiten wie die Datenbankskommunikation oder HTTP-Request-Response-Handling von den genutzten Frameworks übernommen wurden. Mit einer klar vordefinierten Struktur helfen die Frameworks auch dabei, die Wartbarkeit und Erweiterbarkeit der Anwendung zu erhöhen.

6 Fazit

Ergebnisse und Erkenntnisse, die während der Ausarbeitung der Arbeit entstanden sind, werden in diesem Kapitel zusammengefasst und danach wird ein Ausblick über die Weiterentwicklungs- sowie Weiterforschungsmöglichkeiten gegeben.

6.1 Zusammenfassung

Der theoretische Teil dieser Arbeit beschäftigte sich intensiv mit Visualisierungsmöglichkeiten von Unternehmensarchitektur. Nachdem die grundlegenden Begrifflichkeiten erklärt wurde, befasste sich die Arbeit mit der Modellierungssprache ArchiMate, die einen einheitlichen und standardisierten Methode zur Modellierung und Visualisierung der Unternehmensarchitektur bietet. Jedoch findet ArchiMate in der Praxis noch keine verbreitete Verwendung [18]. Verschiedene Ansätze zur Visualisierung von Unternehmensarchitektur wurden basierend auf Forschungs- und Untersuchungsergebnisse gezeigt. Es ergab sich daraus, dass Visualisierungen von sowohl qualitativen als auch quantitativen EA-Daten praxisrelevant sind.

Darauf aufbauend ließen sich die Funktionalitäten der entwickelten Anwendung identifizieren. Der praktische Teil dieser Arbeit verfolgte das Ziel, Daten der Unternehmensarchitektur, die auf ArchiMate basieren, qualitativ sowie quantitativ zu visualisieren. Dafür wurde zunächst ein EA-Metamodell eingeführt, das alle Aspekte von ArchiMate abbilden kann. Die Beziehungen zwischen Elementen in ArchiMate ließen sich danach mit dem vorgestellten Algorithmus analysieren und visualisieren. Es zeigte sich dabei, dass Graphentheorie viel Potenzial für die Visualisierung von Unternehmensarchitektur hat.

Die technische Realisierung der Arbeit verfolgte einen modularisierten Ansatz, wobei die Kommunikation zwischen Client und Server über REST-Schnittstellen erfolgen. Mithilfe der Software Frameworks *Spring* und *AngularJS* wurde die Anwendung implementiert. Der Einsatz von Software Frameworks hat den Vorteil, dass sich der Entwickler weniger um die technische Rahmenbedingung kümmern muss und somit mehr auf die Implementierung der Anwendungslogik konzentrieren kann.

6.2 Ausblick

Die während der Ausarbeitung der Arbeit entstandenen Ideen werden im Folgenden kurz beschrieben.

Die entwickelte Anwendung bietet derzeit nur zwei prototypische Visualisierungsarten an. Um eine umfangreichere Sicht auf die EA-Daten zu ermöglichen, sollen zunächst weitere Visualisierungsarten unterstützt werden. Die in [10] vorgeschlagenen Best-Practices-Visualisierungen können als Vorlage dazu dienen. Quantitative EA-Daten können mittels gängigen Diagrammtypen, beispielsweise Liniendiagramm, Kreisdiagramm, grafisch dargestellt werden. Es lässt sich beobachten, dass diese Diagrammtypen von vielen viele Software Bibliothek sowie Frameworks, z. B. d3[9], unterstützt werden. Dagegen findet in der Praxis noch keine programmatische Unterstützung für die typischen Visualisierungsarten im EAM-Bereich, beispielsweise Bebauungsplangrafik, Clustergrafik, Informationsflussgrafik. Sinnvoll wäre es, eine JavaScript-Bibliothek zu entwickeln, die die Erstellung und Bearbeitung der vielfältigen Visualisierungsarten im EAM-Bereich ermöglicht.

Des Weiteren stellt die entwickelte Anwendung nur eine statische Sicht auf die Unternehmensarchitektur dar. Das heißt, dass der Benutzer die EA-Daten lesen, jedoch nicht bearbeiten kann. Im Fall eines Änderungswunschs der EA-Daten muss er auf andere Werkzeuge oder Ressourcen zugreifen. Somit entsteht der Bedarf an eine Interaktive Anwendung mit dem Architekturmodell. Beispielsweise kann der Benutzer die Zellen einer Bebauungsplangrafik schieben oder entfernen, um die Beziehung zwischen einer Anwendung und eines Geschäftsprozesses zu ändern. Dementsprechend soll die EA-Daten und somit das darunterliegende Architekturmodell aktualisiert werden. Hier muss allerdings auf Datenkonsistenz und Datenqualität geachtet werden.

[9] https://d3js.org/ Zugriff am [31.05.2017]

Literaturverzeichnis

[1] International Organization for Standardization, „ISO/IEC/IEEE 42010:2011, Systems and software engineering - Architecture description," Geneva, 2011.

[2] The Open Group, „TOGAF® Version 9.1," 2011. [Online]. Available: http://pubs.opengroup.org/architecture/togaf9-doc/arch/. [Zugriff am 31. 05. 2017].

[3] R. E. Giachetti, Design of enterprise systems: theory, architecture, and methods, CRC Press, 2010.

[4] J. H. Keuntje und R. Barkow, Hrsg., Enterprise Architecture Management in der Praxis: Wandel, Komplexität und IT-Kosten im Unternehmen beherrschen, Symposion Publishing GmbH, 2010.

[5] M. Lankhorst, Enterprise architecture at work: modelling, communication and analysis, 3. Hrsg., Springer-Verlag Berlin Heidelberg, 2013.

[6] F. Ahlemann, E. Stettiner, M. Messerschmidt und C. Legner, Hrsg., Strategic enterprise architecture management: challenges, best practices, and future developments, Springer Science & Business Media, 2012.

[7] W. Keller, IT-Unternehmensarchitektur: Von der Geschäftsstrategie zur optimalen IT-Unterstützung, dpunkt Verlag, 2007.

[8] The Open Group, „ArchiMate® 3.0 Specification," The Open Group, 2016.

[9] The Open Group, „ArchiMate® Model Exchange File Format for the ArchiMate Modeling Language, Version 3.0," The Open Group, 2017.

[10] I. Hanschke, Enterprise Architecture Management - einfach und effektiv: Ein praktischer Leitfaden für die Einführung von EAM, 2. Hrsg., Carl Hanser Verlag München, 2016.

[11] H. Jonkers, I. Band, D. Quartel und M. Lankhorst, „ArchiSurance case study," The Open Group, 2016.

[12] A. Wittenburg, „Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften," Ph.D. Dissertation, Fakultät für Informatik, Technische Universität München, 2007.

[13] S. Kruse, J. S. Addicks, M. Postina und U. Steffens, „Decoupling models and visualisations for practical EA tooling," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, 2009. S. 62-71.

[14] S. Roth, M. Hauder, M. Zec, A. Utz und F. Matthes, „Empowering business users to analyze enterprise architectures: structural model matching to configure visualizations," in *17th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, 2013. S. 352-360.

[15] I. B. Monahov, „Integrated software support for quantitative models in the domain of enterprise architecture management," Ph.D. Dissertation, Fakultät für Informatik, Technische Universität München, 2014.

[16] M. B. W. Schätzlein, „A generic and integrated software support for the visualization of metrics in the context of Enterprise Architecture Management," Master's Thesis in Wirtschaftsinformatik, Fakultät für Informatik, Technische Universität München, 2014.

[17] F. Matthes, I. Monahov, A. Schneider und C. Schulz, „EAM KPI Catalog v 1.0," Technische Universität München, 2012.

[18] S. Roth, M. Zec und F. Matthes, „Enterprise architecture visualization tool survey 2014," Technische Universität München, 2014.

[19] I. Sommerville, Software engineering, 9. Hrsg., Addison-Wesley, 2011.

[20] B. Bruegge und A. H. Dutoit, Object-oriented software engineering using UML, patterns, and Java, 3. Hrsg., Prentice Hall, 2010.

[21] R. Sedgewick und K. Wayne, Algorithms, 4. Hrsg., Addison-Wesley Professional, 2011.

[22] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures," Ph.D. Dissertation, University of California, 2000.

Abbildungsverzeichnis

Tabellenverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____