



Digitalisierung mit APIs:
Möglichkeiten zur massenhaften API-fizierung im Kontext der
digitalen Transformation mithilfe eines Modells
zum automatisierbaren Design von Systemintegrationsarchitekturen

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

Angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von M.Sc. Konrad Nadobny

geb. am 05.02.1993 in Berlin

Gutachter

Prof. Dr. Andreas Schmietendorf

Prof. Dr. Klaus Turowski

Prof. Dr. Rüdiger Zarnekow

Magdeburg, den 07.02.2025

Zusammenfassung

Diese Arbeit beleuchtet die Herausforderungen bei der digitalen Transformation in Bezug auf Systemintegration. In diesem Kontext werden Möglichkeiten zur massenhaften API-fizierung untersucht. Im Ergebnis erarbeitet der Autor ein Bewertungsmodell, mit welchem das Design von Integrationsarchitekturen automatisiert gestaltet werden kann. Dieses als GAP¹-Modell bezeichnete Konzept wird abschließend anhand von Fallstudien validiert.

Die Arbeit gliedert sich in drei aufeinander aufbauende Teile, welche durch Einleitung und Ausblick ergänzt werden:

- **Analyse:** Anforderungsanalyse und Definition einer Taxonomie anhand analytischer Literaturrecherche und eigener vorhergehender Forschungsergebnisse
- **Synthese:** Konzeption und Parametrisierung eines Bewertungsmodells
- **Validierung:** Empirische Untersuchung anhand von Fallstudien

Die Arbeit ist eingebettet in das übergeordnete Themenfeld der digitalen Transformation. Das Institut für Deutsche Wirtschaft (IW) schreibt hierzu:

Die Kennzeichen der Digitalisierung sind die Virtualisierung und Vernetzung der realen Welt, das Teilen von Daten sowie die plattformbasierte Organisation von Wertschöpfungsketten. Das Besondere daran ist, dass Daten und Datenmodelle keinem physischen Verschleiß unterliegen und deshalb von mehreren Akteuren gleichzeitig und mehrfach genutzt werden können. Dies eröffnet zugleich eine hohe Skalierungsfähigkeit von Geschäftsmodellen und deren Organisation über Plattformen. [2]

Die Digitale Transformation bedeutet für das Fachgebiet der IT-Architektur, dass historisch gewachsene Systemlandschaften zu digitalen Konnektivitätsökosystemen weiterentwickelt werden müssen. In diesen modernen Konnektivitätsökosystemen werden die Daten und Funktionalitäten verschiedener Systeme über APIs² verbunden und orchestriert. Dies ermöglicht, dass Prozesse und Abläufe hoch automatisiert ohne Systembrüche realisiert werden können.

Grundlage für das Gelingen der digitalen Transformation ist Kompetenzsouveränität im Bereich der Integrationsarchitektur. Für eine massenhafte API-fizierung³ fehlt es jedoch an genügend Fachkräften [4, Seite 38]. Aus dieser Motivation heraus forscht der Autor im Rahmen dieser Arbeit nach Möglichkeiten, wie diese Mangellage kompensiert werden kann. Dabei entsteht die Idee, dass ein automatisierbarer modellbasierter Ansatz zum Design von Integrationsarchitekturen eine Möglichkeit ist, um die Effizienz in der IT-Architektur zu verbessern.

Der Autor formuliert die These, dass sich Integrationsarchitekturen modellgetrieben optimieren lassen. Den Kontext hierfür gibt unter Anderem das „The Open Group Architecture Framework“ (TOGAF), welches das führende Framework für Enterprise Architektur ist. [5]

Grundlage für die Optimierung von Integrationsarchitekturen ist ein entsprechendes Messkonzept. Die Klassifizierung von Integrationsarchitekturen ist dabei der erste Schritt, um diese miteinander vergleichen zu können. Ausgehend vom aktuellen Stand der Forschung entwickelt der Autor daher eine Taxonomie, welche es erlaubt Integrationsarchitekturen zu Segmentieren und anhand ihrer Merkmale zu

¹ GAP steht für „Good Architecture Practice“ und ist angelehnt an den GxP-Begriff qualifizierter Computersysteme [1].

² APIs sind die Elemente einer Systemlandschaft, welche für die Integration der verschiedenen Systeme genutzt werden (vgl. Systemschnittstellen, „Kleber zwischen den Systemen“) [22]

³ API-fizierung beschreibt den Transformationsprozess dazu, dass Daten und Funktionalitäten über standardisierte und maschinenlesbare Zugriffsschicht in einem Konnektivitätsökosystem bereitgestellt werden. [3]

klassifizieren. Die vertikale Segmentierung erfolgt dabei auf Grundlage der in TOGAF definierten Granularitätsebenen von IT-Architektur. Die horizontale Segmentierung erfolgt dann jeweils pro Granularitätsebenen als Klassifizierung von Designartefakten anhand vorher definierter Merkmale. Im Ergebnis kann jeder Integrationsarchitektur einem bestimmten Architekturarchetypen zugeordnet werden, welcher sich aus der Kombination der Klassifizierung auf den folgenden drei Granularitätsebenen ergibt:

Die **Enterprise-Ebene** betrachtet Integrationslandschaften im größten Detailgrad und enthält die Gesamtheit aller IT-Systeme und -Anwendungen innerhalb einer Organisation. [6, Seite 2] Eine Klassifizierung zu Topologien erfolgt anhand der Merkmale Kontrollstrategie (zentral, oder verteilt) und Kommunikationsflexibilität (exklusiv, oder distributiv). [7] Ausprägungen sind *Integration-Hub*, *-Bus*, *-Tree* und *-Mesh*. [8]

Die **Applikationsebene** beschreibt die Integrationsarchitektur zwischen einzelnen Systemen, welche wiederum aus mehreren Komponenten bestehen können. [9, Seite 21] Eine Klassifizierung nach Paradigmen erfolgt auf Grundlage von Initialisierung des Informationsaustauschs (proaktiv, oder reaktiv), sowie der Ausführung der Informationsintegration (materialisiert, oder virtualisiert) [10] Hieraus ergeben sich die Klassen *Storage-Driven*-, *Service-Driven*- und *Event-Driven*-Integration.

Die **Technische Ebene** erfasst die technische Implementierung einer bestimmten Komponente innerhalb der Systemlandschaft. [9, Seite 20–25] Eine Klassifizierung nach Methoden erfolgt auf Grundlage der Merkmale Formats der Datenendeinrichtung (dateibasiert, webbasiert, bzw. tabellenbasiert), sowie Kommunikationsart der Übertragungseinrichtung (push. bzw. pull). [11] Hieraus ergeben sich die Integrationsmethoden *File-Exports* und *-Imports*, *Webhooks* und *Webservices*, sowie *Change-Data-Capture* und *Database-Connection*. Als weiteres Unterscheidungsmerkmal dient die Systemnativität der Übertragungseinrichtung, welche eine ergänzende Einordnung als *Native API*, bzw. *Wrapper/Mediator*-Integration erlaubt. [12]

Ein direktes Kopieren von Daten über eine Datenbankschnittstelle kann somit als *Storage-Driven-Integration* mit *Database-Connection* in einem *Integration-Mesh* kategorisiert werden.

Eine Bewertung der verschiedenen Integrationsarchitekturen erfolgt auf einer Ordinalskala in sieben Dimensionen nach den Bewertungskriterien *Robustheit*, *Integrität*, *Skalierbarkeit*, *Kontrolle*, *Flexibilität*, *Durchsatz* und *Latenz*. Die Architekturbewertung ergibt sich dabei aus der Kombination der Stärken und Schwächen der jeweiligen Topologie, des Paradigmas, sowie der Methode, welche mit dem individuellen Anforderungsprofil abgeglichen werden.

Ausgehend von der Konzeptionalisierung des GAP-Modells ermittelt der Autor anhand einer vergleichenden Bewertung der verschiedener Architekturmerkmale Wertzahlen auf allen Dimensionen für alle definierten Kategorien. Diese Wertzahlen werden zur initialen Parametrisierung des Bewertungsmodells benötigt, welches dann im nächsten Schritt empirisch validiert wird.

Die Validierung dieses Ansatzes erfolgt anhand von drei Fallstudien mit verschiedenen implementierten Integrationsarchitekturen. Eine empirische Plausibilisierung der Ergebnisse des Bewertungsmodells erfolgt dabei anhand des Vergleiches mit Ergebnissen eines unabhängigen Application Assessments. Diese ganzheitliche Untersuchung erfolgt regelmäßig als Teil des Lebenszyklusmanagements im Kontext der Enterprise Architektur. Da hierbei jedoch die jeweiligen Applikationen und nicht speziell die Integrationen zwischen diesen betrachtet werden, werden die vorliegenden Ergebnisse durch die Beschreibung der tatsächlichen Implementierung kontextualisiert.

Im Ergebnis zeigen die Fallstudien, dass das GAP-Modell korrekte Ergebnisse liefert und somit ein zweckdienliches Werkzeug für IT-Architekten ist. Mithilfe dieses Ansatzes kann einerseits Handlungsbedarf in bestehenden Integrationsarchitekturen sehr einfach ermitteln und andererseits werden IT-Architekten in der Auswahl der passenden Integrationsarchitektur durch den modellgetriebenen Ansatz enorm entlastet, so dass sie sich auf andere Aufgaben, bspw. auf die tatsächliche Umsetzung der

Integration konzentrieren können. Es unterstützt somit die Skalierung der digitalen Transformation und stärkt die Fähigkeit zur massenhaften API-fizierung von Systemlandschaften.

Potential zur Optimierung und Weiterentwicklung identifiziert der Autor insbesondere bei der weiteren Ausarbeitung der Architekturbewertungen, wobei bereits die vorliegende Parametrisierung klare und plausible Ergebnisse liefert.

Auch in Bezug auf die weitere Automatisierung und Vereinfachung der Anwendung des Modells gibt es Ausbaumöglichkeiten. So erfordert derzeit die Anwendung des GAP-Modells noch ein fundiertes Fachwissen und Fähigkeiten bei der Identifikation von Mustern bei den Designartefakten. Potenzial liegt hierbei insbesondere in der KI-gestützte Automatisierung, beispielsweise zur Mustererkennung, sowie Analyse bestehender Implementierungen und Generierung von Integrationscode.

Abschließend ist auch die Erweiterung des GAP-Modells auf zusätzliche tiefere Granularitätsebenen denkbar und würde noch differenziertere Aussagen ermöglichen, indem beispielsweise Best-Practices und konstruktive Designmuster auf APIs verschiedener Methoden Anwendung finden. Somit wäre nicht nur eine Messung und Optimierung der Architektur, sondern auch der jeweiligen Implementierungen möglich.

Abstract

This thesis pulls a spotlight on the challenges of digital transformation in terms of system integration. In this context, the author develops an evaluation model with which the design of integration architectures can be automated to enable the massive API-fication of legacy applications. This approach is named “GAP model” and is being validated through three case studies that are described in the later part of this thesis.

The work is divided into three consecutive parts, which are supplemented by an introduction and an outlook:

- **Analysis:** Requirements analysis and definition of a taxonomy based on analytical literature research and our own previous research results
- **Synthesis:** Conceptualization and parameterization of an evaluation model
- **Validation:** Empirical investigation based on case studies

The Institute for German Economy (IW) states the following about Digital Transformation, that constitutes the wider frame for this thesis:

The key characteristics of digitalization are the virtualization and networking of the real world, the sharing of data and the platform-based organization of value chains. As data and data models do not physically decay, they can be used by several actors simultaneously and multiple times, which opens up a high level of scalability of business models and their organization via platforms. [2]

In context of IT architecture, the challenge of digital transformation is to transform historically grown system landscapes into modern connected digital ecosystems. In these modern ecosystems the data and functionality of different systems is connected and orchestrated via APIs. This enables processes and procedures to be implemented in a highly automated manner without system disruptions.

The basis for the success of digital transformation is competence sovereignty in integration architecture. However, there is a severe lack of sufficient specialists for mass API-fication [4, Seite 38]. Therefore, the author is looking for ways to address this shortcoming. This results in the idea of an automatable model-based approach towards the design of integration architectures to improve efficiency in IT architecture. The Author embeds this idea of model-driven integration architecture optimization into the established “The Open Group Architecture Framework” (TOGAF), which is the leading framework for enterprise architecture.[5]

In order to optimize integration architectures, an appropriate measurement concept is imperative. The classification of integration architectures is the first step in order to be able to compare them with each other. Based on the current state of research, the author develops a taxonomy that allows integration architectures to be segmented and classified based on their characteristics. The vertical segmentation is based on the granularity levels of IT architecture defined in TOGAF. The horizontal segmentation is then carried out per each level. For this, a classification of design artifacts based on previously defined characteristics is being applied. As a result, each integration architecture can be assigned to a specific architectural archetype, which results from the combination of the classification at the following three levels of granularity:

The **Enterprise Architecture** level focusses on high-level integration landscapes and describes the entirety of all IT systems and applications within an organization. [6, Seite 2] Topologies are classified based on the characteristics of control strategy (central, or distributed) and communication flexibility (exclusive, or distributive).[7] The resulting classification determines topologies as *Integration-Hub*, *-Bus*, *-Tree* und *-Mesh*. [8]

The **Application Architecture** level describes the integration architecture between individual systems, while those can consist of several components. [9, Seite 21] The classification of respective integration designs into paradigms is based on the initialization of information exchange (proactive or reactive) and the execution of information integration (materialized or virtualized). [10] This results in a classification into *Storage-Driven*-, *Service-Driven*- und *Event-Driven*-Integrations.

The **Technical Architecture** level shows the technical implementation of a specific component within the system landscape. [9, Seite 20–25] The classification into integration methods is based on the characteristics of the data format (file-based, web-based, or table-based), as well as the transmission mode (push or pull). [11] Following this logic, integration methods can be classified as *File-Exports* and *-Imports*, *Webhooks* and *Webservices*, as well as *Change-Data-Capture* und *Database-Connection*. Another distinguishing feature is the system naiveness of the interface, which is a supplementary attribute and allows further classification into *Native API* and *Wrapper/Mediator*-Integrations. [12]

Following this taxonomy, the direct copying of data via a database interface can be categorized as storage-driven integration with database connection in an integration mesh.

The various integration architectures are being evaluated on an ordinal scale in seven dimensions: *robustness*, *integrity*, *scalability*, *control*, *flexibility*, *throughput* and *latency*. The architecture evaluation results from the combination of the strengths and weaknesses of the respective topology, the paradigm, and the method, which are compared with the individual architecture requirements.

In preparation of the empirical validation of the GAP model, the author performs a comparative assessment of all architecture classes on each of the three levels. The outcomes of this assessment are being used as initialization parameters for the use of the GAP model in the case studies.

The author performs three case studies with different implemented integration architectures to validate the GAP-model. An empirical plausibility check of the results of the evaluation model is performed by comparing them with the results of an independent application assessment. This application assessment is a part of the regular life cycle management activities in the context of enterprise architecture. A limiting factor is, that the application assessment measures only the applications and does not contain dedicated information regarding the integration between different systems. However, the results can be contextualized by combining them with contextual insights from the actual implementation, so that the outcomes of this analysis can be applied to the validation of the GAP-model.

The case studies show that the GAP model provides correct results and is therefore a useful tool for IT architects. This approach can help to easily identify the need for action in existing integration architectures. In addition, this model-driven approach drastically reduces the burden for IT architects, as it guides them to choose the appropriate integration architecture. This enables them to concentrate on other more value adding tasks, for example on the actual can concentrate on implementing the integration. The GAP-model therefore is a good tool to enable a large-scale digital transformation and strengthens the ability to mass-APIfy system landscapes.

After the validation, the author identifies potential for optimization and further development. Particularly the further elaboration of the architecture assessments could be further refined, whilst the existing parameterization already providing clear and plausible results.

There are also opportunities for expansion in terms of further automating and simplifying the application of the model. The application of the GAP model currently requires in-depth specialist knowledge and skills in identifying patterns in design artifacts. AI-supported automation would be an enormous additional optimization lever. Features like pattern recognition, as well as analysis of existing implementations and generation of integration code could be of large additional value for architects. Last but not least, extending the GAP model to additional, deeper levels of granularity is also conceivable and would enable even more differentiated statements, for example by applying best practices and constructive design patterns to APIs of various methods. This would not only allow measurement and optimization of the architecture, but also of the respective implementations.

Inhaltsverzeichnis

1. Einführung.....	1
1.1. Motivation und Hintergrund.....	1
1.1.1. Digitalisierung und Digitale Transformation.....	1
1.1.2. Digitalisierung von Geschäftsprozessen in der Klinischen Forschung	2
1.1.3. Massenhafte API-fizierung historisch gewachsener Systemlandschaften	3
1.2. Zielstellung und Forschungsfragen	4
1.3. Wissenschaftliche Methodik und Thematische Einordnung	5
2. Begriffsbestimmung und Einordnung	8
2.1. Digitalisierung.....	8
2.1.1. Digitale Transformation	8
2.1.2. Systemintegration	9
2.1.3. Konnektivitätsökosystem	9
2.1.4. Akteure im Konnektivitätsökosystem.....	10
2.2. API-basierte Systemintegration	12
2.2.1. Application Programming Interface (API).....	12
2.2.2. API-fizierung.....	12
2.2.3. API-Management.....	14
2.2.4. API-Management-Plattform.....	15
2.3. IT-Architektur.....	16
2.3.1. Enterprise-Ebene zur Einordnung von Integrationslandschaften	18
2.3.2. Applikationsebene zur Einordnung von Integrationsmustern	19
2.3.3. Technische Ebene zur Einordnung von Integrationsverfahren	20
2.4. Modellbasierte Ansätze zur Optimierung von Integrationsarchitekturen	21
2.4.1. Standardisierte Notation der Architektur mit UML.....	21
2.4.2. Standardisiertes Management der IT-Architektur mit dem TOGAF Content Framework	23
2.4.3. Standardisierte pArchitekturentwicklung mit der TOGAF Architecture Development Method.....	24
2.4.4. Standardisierte Architekturentscheidungen mit der TOGAF Architecture Alternatives Method.....	25
2.5. Zusammenfassung	26
3. Stand der Forschung und Analyse existierender Ansätze zur Klassifizierung von Integrationsarchitekturen	28
3.1. Beschreibung des analytischen Vorgehens	28
3.2. Enterprise-Ebene: Klassifizierung von Integrationslandschaften nach Integrationstopologie	29
3.2.1. Integration-Hub	30
3.2.2. Integration-Bus.....	30
3.2.3. Integration-Tree	31
3.2.4. Integration-Mesh.....	31
3.3. Applikationsebene: Klassifizierung von Integrationsmustern nach Integrationsparadigma.....	32
3.3.1. Storage-Driven-Integration.....	34
3.3.2. Service-Driven-Integration.....	36
3.3.3. Event-Driven-Integration	37

3.4. Technische Ebene: Klassifizierung von Integrationsverfahren nach Integrationsmethode	40
3.4.1. File-Exports	42
3.4.2. File-Imports	42
3.4.3. Webhooks	42
3.4.4. Webservices	43
3.4.5. Change-Data-Capture.....	43
3.4.6. Database-Connection.....	43
3.4.7. Wrapper/Mediator.....	44
3.5. Zusammenfassung: Definition der Taxonomie	46
3.5.1. Enterprise-Ebene: Klassifizierung nach Integrationstopologie	46
3.5.2. Applikationsebene: Klassifizierung nach Integrationsparadigma	47
3.5.3. Technische Ebene: Klassifizierung nach Integrationsmethode	47
4. Konzeption des GAP-Modells zur Analyse und Bewertung von Integrationsarchitekturen	48
4.1. Anforderungen an ein Modell für Architekturentscheidungen	48
4.1.1. Automatisierbarkeit für den massenhaften Einsatz	49
4.1.2. Einfache Anwendung und Klarheit der Empfehlungen	49
4.1.3. Generische Bewertungsmodell zur Bestandsanalyse.....	49
4.1.4. Modellbasierte Unterstützung von Designentscheidungen.....	49
4.2. Definition von Bewertungskriterien	50
4.2.1. Robustheit	50
4.2.2. Integrität.....	51
4.2.3. Skalierbarkeit.....	51
4.2.4. Kontrolle	51
4.2.5. Flexibilität	52
4.2.6. Durchsatz	52
4.2.7. Latenz	52
4.3. Bewertung von Integrationsarchitekturen.....	53
4.3.1. Konfiguration des individuellen Anforderungsprofils durch Gewichtung von Bewertungskriterien...	54
4.3.2. Parametrisierung des Bewertungsmodells durch Bewertung von Designartefakten	56
4.3.3. Aufbau des Optionsraums durch Kombination klassifizierter Designartefakte.....	58
4.3.4. Bewertung der Architekturarchetypen im Optionsraum durch Anwendung des Bewertungsmodells	58
4.3.5. Modellbasierte Architekturentscheidungen durch Auswertung der Bewertungsergebnisse	60
4.4. Einordnung in das TOGAF Architekturrahmenwerk	62
4.5. Zusammenfassung: Das GAP-Modell in Kürze	64
5. Parametrisierung des GAP-Modells durch vergleichende Bewertung von Designartefakten ..	66
5.1. Enterprise-Ebene: Gegenüberstellung ausgewählter Integrationstopologien	66
5.1.1. Integration-Hub	67
5.1.2. Integration-Bus	68
5.1.3. Integration-Tree	69
5.1.4. Integration-Mesh	70
5.2. Applikationsebene: Gegenüberstellung ausgewählter Integrationsparadigmen.....	71
5.2.1. Storage-Driven-Integration	72
5.2.2. Service-Driven-Integration	73
5.2.3. Event-Driven-Integration.....	74

5.3. Technische Ebene: Gegenüberstellung ausgewählter Integrationsmethoden	75
5.3.1. File-Exports	76
5.3.2. File-Imports	77
5.3.3. Webhooks	78
5.3.4. Webservices	79
5.3.5. Change-Data-Capture	80
5.3.6. Database-Connection	81
5.3.7. Wrapper/Mediator	82
5.4. Zusammenfassung der Bewertungsergebnisse	82
5.4.1. Integrationstopologien	82
5.4.2. Integrationsparadigmen	84
5.4.3. Integrationsmethoden	86
6. Empirische Validierung des GAP-Modells anhand von Fallstudien	88
6.1. Versuchsaufbau	88
6.1.1. Vorgehensweise	89
6.1.2. Kriterien des Application Assessments	90
6.2. Anbindung verschiedener Systeme an ein operatives Data Warehouse	92
6.2.1. Beschreibung der Anforderungen	93
6.2.2. Anwendung des GAP-Modells	94
6.2.3. Beschreibung der tatsächlichen Implementierung	97
6.2.4. Vergleich mit den Ergebnissen des Application Assessments	98
6.3. Anbindung von Satellitensystem an operative Kernsysteme	101
6.3.1. Beschreibung der Anforderungen	102
6.3.2. Anwendung des GAP-Modells	104
6.3.3. Beschreibung der tatsächlichen Implementierung	106
6.3.4. Vergleich mit den Ergebnissen des Application Assessments	109
6.4. Kontinuierliche Synchronisierung von zweier operativer Kernsysteme	115
6.4.1. Beschreibung der Anforderungen	116
6.4.2. Anwendung des GAP-Modells	118
6.4.3. Beschreibung der tatsächlichen Implementierung	121
6.4.4. Vergleich mit den Ergebnissen des Application Assessments	128
6.5. Zusammenfassung und Auswertung	132
6.5.1. Fallstudie 1: Anbindung verschiedener Systeme an ein operatives Data Warehouse	132
6.5.2. Fallstudie 2: Anbindung von Satellitensystem an operative Kernsysteme	132
6.5.3. Fallstudie 3: Kontinuierliche Synchronisierung von zweier operativer Kernsysteme	134
6.5.4. Fazit	135
7. Bewertung und Ausblick	136
7.1. Bewertung der Forschungsergebnisse	136
7.1.1. Reflektion der Forschungsfragen	136
7.1.2. Potential zur Optimierung und weiteren Vertiefung	137
7.2. Ausblick	138
7.2.1. KI-gestützte Automatisierung des GAP-Modells	138
7.2.2. Spezifikationsbasierte Automatisierung	139
7.2.3. Demokratisierung der Softwareentwicklung	140
7.2.4. Flexibilisierung mit domänenspezifischen Datenmodellen	141
8. Quellenverzeichnis	142

Tabellenverzeichnis

<i>Tabelle 1: Übersicht der Klassifizierungsoptionen</i>	46
<i>Tabelle 2: Klassifizierung von Integrationslandschaften nach Integrationstopologie (Enterprise-Ebene)</i>	46
<i>Tabelle 3: Klassifizierung von Integrationsmustern nach Integrationsparadigma (Applikationsebene)</i>	47
<i>Tabelle 4: Klassifizierung von Integrationsverfahren nach Integrationsmethode (Technische Ebene)</i>	47
<i>Tabelle 5: Beispielhafte Gewichtungsmatrix</i>	54
<i>Tabelle 6: Anwendungsbereiche der Bewertungskriterien</i>	56
<i>Tabelle 7: Initiale Parametrisierung des GAP-Modells (siehe Kapitel 6)</i>	57
<i>Tabelle 8: Bewertungsmatrix für Designartefakte auf den drei Ebenen der IT-Architektur</i>	59
<i>Tabelle 9: Geeignete Integrationsarchitekturen</i>	60
<i>Tabelle 10: Integrationstopologien im GAP-Modell (Enterprise-Ebene)</i>	66
<i>Tabelle 11: Integrationsparadigmen im GAP-Modell (Applikationsebene)</i>	71
<i>Tabelle 12: Integrationsmethoden im GAP-Modell (Technische Ebene)</i>	75
<i>Tabelle 13: Bewertungsergebnisse des GAP-Modells für Integrationen mit einem operativen Data Warehouse</i>	95
<i>Tabelle 14: Bewertungsergebnisse des GAP-Modells für Integrationen zur Anbindung von Satelliten</i>	104
<i>Tabelle 15: Bewertungsergebnisse des GAP-Modells für Integrationen zur Synchronisierung operativer Kernsysteme</i>	118

Abkürzungsverzeichnis

API	Application Programming Interface
CDC	Change-Data-Capture
DB	Database / Datenbank
EDB	Enterprise Data Bus
ESB	Enterprise Service Bus
ETL	Extract, Transform, Load
FTP	File Transfer Protocol
GAP	Good Architecture Practice
GxP	Good Practices
ODBC	Open Database Connectivity Protocol
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronic Engineers
IoT	Internet of Things
IW	Institut für Deutsche Wirtschaft
KI/AI	Künstliche Intelligenz / Artificial Intelligence
MSA	Microservice Architecture
NBDIF	NIST Big Data Interoperability Framework
NIST	National Institute of Standards and Technology
ODBC	Open Database Connectivity
On-prem	On Premise (in der eigenen Umgebung betriebene Software)
RPC	Remote Procedure Calls
SaaS	Software as a Service
SOA	Service-orientierte Architektur
TOGAF	The Open Group Architecture Framework
WOA	Web-orientierte Architektur

Abbildungsverzeichnis

Anmerkung:

Bei allen Abbildungen handelt es sich um eigene Darstellungen, wenn nicht explizit im Abbildungstitel auf eine andere Urheberschaft verwiesen wird. Aus diesem Grund verzichtet der Autor in dieser Arbeit auf die dedizierte Kennzeichnung von Abbildungen als „eigene Darstellung“.

Abbildung 1: Schematische Darstellung des Aufbaus dieser Arbeit	5
Abbildung 2: Kategorisierung von Akteuren im Konnektivitätsökosystem	10
Abbildung 3: Full-Lifecycle-API-Management-Modell (Quelle: Schmietendorf et al. [43, Seite 39] in Anlehnung an Chiu [44])	14
Abbildung 4: Schematische Darstellung einer API-Management-Plattform (Quelle: Schmietendorf et al. [43])	15
Abbildung 5: TOGAF Content Framework (Quelle: Eigene Darstellung in Anlehnung an The Open Group [48, Seite 5])	16
Abbildung 6: Granularitätsebenen von Informationssintegration (Quelle: Eigene Darstellung in Anlehnung an The Open Group [9, Seite 25])	17
Abbildung 7: Beispielhafte Darstellung einer Integrationslandschaft auf Ebene der Enterprise Architecture	18
Abbildung 8: Beispielhafte Architekturdarstellung auf Ebene der Application-Architecture	19
Abbildung 9: Beispielhafte Darstellung einer Lösungsarchitektur auf technischer Ebene	20
Abbildung 10: UML zur Modellierung von Systemintegration auf Enterprise-Architecture-Ebene	21
Abbildung 11: Vereinfachte Notation von Integrationspfaden	22
Abbildung 12: Einordnung des GAP-Modells in das TOGAF Content Framework (Quelle: Eigene Darstellung in Anlehnung an The Open Group [48, Seite 5])	23
Abbildung 13: Einbettung des GAP-Modells in die Iterationszyklen der TOGAF Architecture Development Method (Quelle: Eigene Darstellung in Anlehnung an The Open Group [9, Seite 1–5])	24
Abbildung 14: Architekturentscheidungsprozess nach TOGAF (Quelle: Eigene Darstellung in Anlehnung an The Open Group [9, Seite 74])	25
Abbildung 15: Kategorisierung von Integrationstopologien	29
Abbildung 16: Informationspfad in einer Hub-Topologie	30
Abbildung 17: Informationspfad in einer Bus-Topologie	30
Abbildung 18: Informationspfad in einer Tree-Topologie	31
Abbildung 19: Informationspfad in einer Mesh-Topologie	31
Abbildung 20: Klassifizierung von Integrationsarchitekturmustern	33
Abbildung 21: Beispielhafte Klassifizierung von Integrationsparadigmen	33
Abbildung 22: schematische Darstellung eines ETL-Prozesses	35
Abbildung 23: Data Hub als Beispiel für Storage-Driven-Integration	35
Abbildung 24: Schematische Darstellung einer Servicebasierten Integrationsprozesses	36
Abbildung 25: Schematische Darstellung eines eventbasierten Integrationsprozesses	38
Abbildung 26: Schematische Darstellung eines eventbasierten Integrationsprozesses mit Event-Queue	39
Abbildung 27: IoT Connected Car als Beispiel für Event-Driven-Integration	39
Abbildung 28: Datenübertragung nach DIN 44302 (Quelle: Eigene Darstellung in Anlehnung an DIN 44302 [61])	40
Abbildung 29: Klassifizierung von Integrationsmethoden	41
Abbildung 30: Beispielhafte Klassifizierung von Integrationsmethoden	41
Abbildung 31: Schematische Darstellung eines API-Wrappers	44
Abbildung 32: Beispielhafte Orchestrierung von Microservices in einer Middleware	45
Abbildung 33: Vorgehen bei der Bewertung von Integrationsarchitekturen unter Anwendung des GAP-Modells	53
Abbildung 34: Berechnung der Ziel-Bewertung des Anforderungsprofils	55
Abbildung 35: Beispielhaftes Anforderungsprofil	55
Abbildung 36: Architekturarchetyp „Bus-Topologie nach Storage-Driven-Paradigma mit Database-Connection“	58
Abbildung 37: Beispielhafte Visualisierung der GAP-Modell Empfehlung	61

Abbildung 38: Einbettung des GAP-Modells in die TOGAF Architecture Alternatives Method.....	62
Abbildung 39: Gegenüberstellung ausgewählter Integrationstopologien	83
Abbildung 40: Gegenüberstellung ausgewählter Integrationsparadigmen.....	85
Abbildung 41: Gegenüberstellung ausgewählter Integrationsmethoden	87
Abbildung 42: Versuchsaufbau zur empirischen Validierung des GAP-Modells	88
Abbildung 43: Vorgehen und Ergebnisse von Fallstudie 1	92
Abbildung 44: Anforderungsprofil an Integrationen für ein operatives Data Warehouse.....	93
Abbildung 45: Integrationsarchitekturen für Fallstudie 1 mit verschiedenen Paradigmen und Methoden im Vergleich	96
Abbildung 46: Implementierung des operativen Data Warehouses im Fallbeispiel	97
Abbildung 47: Auswertung der Ergebnisse für ein operatives Data Warehouse	98
Abbildung 48: Ergebnisse des Application Assessments für das operative Data Warehouse (Quelle: Eigener Screenshot aus einem internen Expertensystem).....	99
Abbildung 49: Vorgehen und Ergebnisse von Fallstudie 2	101
Abbildung 50: Anforderungsprofil an Integrationen zur Anbindung von Satelliten	102
Abbildung 51: Integrationsarchitekturen für Fallstudie 2 mit verschiedenen Paradigmen im Vergleich	105
Abbildung 52: Implementierung der Integration von Satellitensystemen	106
Abbildung 53: Ergebnisse für Integrationen zur Anbindung von Satelliten	109
Abbildung 54: Application Assessment für das operative Kernsystem (Quelle: Eigener Screenshot aus einem internen Expertensystem)	110
Abbildung 55: Ergebnisse des Application Assessment für Satellit 1 (Quelle: Eigener Screenshot aus einem internen Expertensystem)	111
Abbildung 56: Ergebnisse des Application Assessment für Satellit 2 (Quelle: Eigener Screenshot aus einem internen Expertensystem)	112
Abbildung 57: Ergebnisse des Application Assessment für Satellit 3 (Quelle: Eigener Screenshot aus einem internen Expertensystem)	113
Abbildung 58: Ergebnisse des Application Assessment für Satellit 4 (Quelle: Eigener Screenshot aus einem internen Expertensystem)	114
Abbildung 59: Vorgehen und Ergebnisse von Fallstudie 3	115
Abbildung 60: Anforderungsprofil an Integrationen zur Synchronisierung operativer Kernsysteme	116
Abbildung 61: Integrationsarchitekturen für Fallstudie 3 mit verschiedenen Topologien und Methoden im Vergleich	119
Abbildung 62: schematische Darstellung tatsächlich implementierter Integrationen in Fallstudie 3	121
Abbildung 63: Ablauf der Synchronisierung mit File-Exports in Fallstudie 3	122
Abbildung 64: Ablauf der Synchronisierung mit Webservices in Fallstudie 3.....	124
Abbildung 65: Ablauf der Synchronisierung mit Events in Fallstudie 3.....	126
Abbildung 66: Ergebnisse für Integrationen zur kontinuierlichen Synchronisierung operativer Kernsysteme	128
Abbildung 67: Ergebnisse des Application Assessment für Integration mit File-Exports (Quelle: Eigener Screenshot aus einem internen Expertensystem).....	129
Abbildung 68: Ergebnisse des Application Assessment für Integration mit Webservices (Quelle: Eigener Screenshot aus einem internen Expertensystem).....	131

**In einer vernetzten Welt werden Informationen
und die damit verbundenen Prozesse,
Systeme und Netzwerke
zu kritischen Wirtschaftsgütern.**

ISO 27000

*In an interconnected world, information
and related processes, systems, and networks
constitute critical business assets. [13, Seite 13]*

1. Einführung

In diesem Kapitel erfolgt eine Einführung in das Thema dieser Arbeit, sowie eine Erläuterung der Struktur und wissenschaftlichen Vorgehensweise.

Der Autor beschreibt zunächst ausgehend vom aktuellen Lagebild zur Digitalisierung in Deutschland vom Institut für Deutsche Wirtschaft die Rolle von API-fizierung und Systemintegration für die digitale Transformation und zeigt auf, wie die fehlende Kompetenzsouveränität die Modernisierung historisch gewachsener Strukturen in Deutschland hemmt. Anschließend erarbeitet der Autor Forschungsfragen und erläutert die wissenschaftliche Methodik sowie Struktur der Arbeit und ordnet das Thema in den Bereich der Wirtschaftsinformatik ein.

1.1. Motivation und Hintergrund

1.1.1. Digitalisierung und Digitale Transformation

Das Institut für Deutsche Wirtschaft (IW) schreibt in seinem Lagebild zur Digitalisierung in Deutschland:

Die Kennzeichen der Digitalisierung sind die Virtualisierung und Vernetzung der realen Welt, das Teilen von Daten sowie die plattformbasierte Organisation von Wertschöpfungsketten. Das Besondere daran ist, dass Daten und Datenmodelle keinem physischen Verschleiß unterliegen und deshalb von mehreren Akteuren gleichzeitig und mehrfach genutzt werden können. Dies eröffnet zugleich eine hohe Skalierungsfähigkeit von Geschäftsmodellen und deren Organisation über Plattformen. [2]

Es ist klar, dass die digitale Transformation entscheidend für unsere wirtschaftliche Zukunft ist. Die Bitkom beschreibt die digitale Transformation als Zusammenspiel der Digitalisierung von Geschäftsmodellen, sowie der Digitalisierung von Geschäftsprozessen. [14] Hierbei spielt in der technologischen Dimension das Feld der Systemintegration eine zentrale Rolle. [15]

Diese Arbeit entstand im Kontext eines Industrieprojektes zur Digitalisierung der klinischen Forschung und Entwicklung der Pharma-Division in einem DAX-Konzern. Im aktuellen Lagebild des IW zur Digitalisierung zeigt sich, dass sich im Betrachtungszeitraum von 2020 bis 2022 der Digitalisierungsindex nur moderat verbessert hat. Die Digitalisierung der Grundstoffe-, Chemie- und Pharmaindustrie ist hierbei durchschnittlich und daher ein guter Referenzpunkt. Gleichzeitig schneidet dieser Industriezweig in Bezug auf Innovation überdurchschnittlich gut ab, wobei das IW in dieser Branche das Digitalisierungspotential als besonders hoch einschätzt. [4, Seite 7–12]

Eine Auswahl relevanter Auszüge aus der Studie „Digitalisierungsindex 2022“ findet sich im Anhang 1.

1.1.2. Digitalisierung von Geschäftsprozessen in der Klinischen Forschung

Grundlage für den wissenschaftlichen und wirtschaftlichen Erfolg forschender Pharmaunternehmen ist eine leistungsfähige und produktive Forschung und Entwicklung. Die Digitalisierung eröffnet hier kontinuierlich neue Möglichkeiten Geschäftsprozesse deutlich zu verbessern. Bereits jetzt hat die klinische Forschung und Entwicklung stark von der stetig voranschreitenden Digitalisierung profitiert und dieser Trend wird sich auch in Zukunft fortsetzen. [16]

Die Leistungsfähigkeit der Klinischen Forschung lässt sich anhand zweier Parameter grob umreißen: Effizient und Effektivität. Dies umfasst sowohl die Fähigkeit einen wirksamen Wirkstoff zu entwickeln (Effektivität), als auch dies in möglichst kurzer Zeit, bzw. zu möglichst geringen Kosten zu realisieren (Effizienz). [17]

Die Effektivität beschreibt den Mehrwert, welcher durch einen neu erforschten und zugelassenen Wirkstoff erzielt wird (Output vs. Outcome). Erfolg kann es nur geben, wenn wirksame Medikamente gefunden werden. Einen besonders hohen Mehrwert für die Patienten generieren innovative Medikamente und Therapien mit verbesserter Wirkung und geringerer Nebenwirkung. [17] Die digitale Transformation steigert die Effektivität, da sich das verfügbare Repertoire an Anwendungen und Therapieansätzen Aufgrund der Digitalisierung dramatisch erweitert.

Effizienz ergibt sich aus dem Verhältnis des investiertem Forschungsaufwands für ein zugelassenes Medikament (Input vs. Output). [17] Die digitale Transformation beeinflusst die Effizienz, da Kosten gesenkt werden und sich die Prozesse beschleunigen. [17] Einerseits fallen für klinische Studien Kosten von mehreren zehntausend Euro pro Tag an, andererseits entstehen mit jedem Tag für das Unternehmen Opportunitätskosten im einstelligen Millionenbereich. Eine frühzeitige Zulassung ermöglicht die Realisierung höherer Vergütung, weil das Medikament früher am Markt angeboten werden kann und somit mehr Zeit bis zum Ende des Patentschutzes für die exklusive Vermarktung des Medikaments bleibt. [18] Technische Innovationen senken dabei die Kosten, eröffnen neue Patientenpotentiale und können vormals langwierige Prozesse durch Automatisierung und Integration skalieren und stark beschleunigen. [19] Ein Beispiel hierfür ist die Rekrutierung von Patienten für eine klinische Studie. Nur wenn in einer möglichst kurzen Zeit eine ausreichende Anzahl geeigneter Studienteilnehmer gewonnen werden können, kann eine ausreichend aussagekräftige Datenbasis zur untersuchten Therapie generiert werden. Eine datengetriebene, automatisierte Identifikation und Auswahl von Medizinischen Einrichtungen mit entsprechenden Patienten kann so einen hohen Beitrag zu einer Effizienteren klinischen Forschung beitragen. [20]

1.1.3. Massenhafte API-fizierung historisch gewachsener Systemlandschaften

Nach einer deutlichen Zunahme der Digitalisierung der Wirtschaft in Deutschland zwischen den Erhebungsjahren 2020 und 2021 wächst diese im Erhebungsjahr 2022 nur marginal. [4]

Die vom Autor betrachtete Ausgangssituation steht exemplarisch für die allgemeine Situation in Deutschland. Zwar werden bereits Digitalisierungspotentiale genutzt, jedoch bremsen fehlende Expertise und begrenzte Ressourcen die digitale Transformation. Diese Kompetenzsouveränität ist jedoch die Grundlage dafür, die digitale Transformation auf breiter Front umzusetzen. [4, Seite 38]

In Bezug auf IT-Architektur führt die fehlende Kompetenzsouveränität zwangsläufig zu Herausforderungen bei der Systemintegration. Historisch gewachsene Systemlandschaften müssen kontinuierlich an die geänderten Anforderungen angepasst werden. Mangelt es an guten Regeln, Normen, Fähigkeiten und Ressourcen, so kann es keine gute Enterprise Architektur geben. An ihre Stelle tritt die „Accidental Architecture“, also eine historisch gewachsene Systemlandschaft ohne spezifische technologische und konzeptionelle Standards. Das Problem dabei ist, dass schlechte Architekturen die Flexibilität des Ökosystems verringern, Integrationsprojekte komplex und zeitaufwendig machen und somit die Agilität des gesamten Unternehmens massiv bremsen. [21, Seite 28–33]

Bei den in den Fallstudien betrachteten Anwendungsfällen sind beispielsweise im Laufe der Zeit unterschiedlichste Paradigmen und Technologien zum Tragen gekommen sind. Zahlreiche Direktverbindungen zwischen den integrierten Systemen und deren inkonsistente Konzeption sorgen für eine hohe Komplexität. Die Implementierungen sind ohne spezifisches Domänenwissen technisch nur noch schwer zu durchdringen. Die technische Heterogenität erweist sich zudem im Betrieb problematisch und führt zu hohen Betriebskosten. Zusätzlich machen versteckte Abhängigkeiten jegliche Anpassung innerhalb der Systemlandschaft äußerst schwierig, wenn diese nicht professionell erfasst und gemanagt werden. Eine ausschnitthafte Darstellung dieser Systemlandschaft findet sich in Anhang 3.

Schlechte Architektur bremst also Agilität und Innovation. Diese ist jedoch grundlegende Voraussetzung für das Gelingen der digitalen Transformation. [22] Aus Sicht der Wirtschaftsinformatik müssen daher Mittel und Wege gefunden werden, wie historisch gewachsene Systemlandschaften unter dem Vorzeichen begrenzter Ressourcen in gut designte Konnektivitätsökosysteme überführt werden können. Hieraus ergibt sich das in Abschnitt 1.2 beschriebene Ziel, ein automatisierbares Modell zum optimalen Design von Systemintegrationen auf einer großen Skala zu erarbeiten.

1.2. Zielstellung und Forschungsfragen

Zentrale These der Arbeit ist, dass sich im Kontext der digitalen Transformation die Systemintegration mithilfe eines automatisierbaren modellgetriebenen Ansatzes auf einer großen Skala optimieren lässt, so dass die Kompetenzsouveränität in diesem Bereich unterstützt wird.

These:

Integrationsarchitekturen lassen sich modellgetrieben optimieren.

Modell zur Optimierung der Systemintegration

Das Ziel dieser Arbeit ist die Erarbeitung eines modellgetriebenen Ansatzes zur automatischen Optimierung von Systemintegrationsarchitekturen. Dabei soll eine Blaupause zur Unterstützung von IT-Architekten bei der digitalen Transformation in Bezug auf Systemintegration entstehen. Es stellt sich dabei die Frage, wie ein solches Modell aussehen kann, wie Integrationsarchitekturen modelliert und bewertet werden können, und wie die Optimierung der Systemintegration als Transformationsprozess in das IT-Management integriert werden kann.

-
- | | |
|---------------|---|
| FF 1.1 | Wie kann Systemintegration modellgetrieben verbessert werden? |
| FF 1.2 | Wie lassen sich Integrationsarchitekturen modellieren? |
| FF 1.3 | Wie lässt sich der Transformationsprozess operationalisieren? |
-

Klassifizierung von Integrationsarchitekturen

Grundlage für einen modellgetriebenen Ansatz ist die Vergleichbarkeit von verschiedenen Systemintegrationen. Es stellt sich daher die Frage, welche Merkmale zur trennscharfen Unterscheidung genutzt werden können.

-
- | | |
|-------------|--|
| FF 2 | Nach welcher Systematik können Integrationsarchitekturen unterschieden werden und welche Klassifizierung ergibt sich daraus? |
|-------------|--|
-

Bewertung von Integrationsarchitekturen

Einer Verbesserung der Systemintegration über die Optimierung des Architekturdesigns liegt die Annahme zugrunde, dass diese Designentscheidungen einen signifikanten Einfluss auf die Qualität von Systemintegration haben. Daher ist zu klären, inwieweit sich unterschiedliche Designentscheidungen auf die Bewertung in Bezug auf die individuellen Anforderungen anhand der definierten Kriterien auswirken.

-
- | | |
|-------------|--|
| FF 3 | Wie lassen sich Integrationsarchitekturen in Bezug auf die individuellen Anforderungen modellbasiert bewerten? |
|-------------|--|
-

1.3. Wissenschaftliche Methodik und Thematische Einordnung

Diese Arbeit beschäftigt sich mit der Digitalisierung mit APIs und beleuchtet Möglichkeiten zur massenhaften API-fizierung im Kontext der digitalen Transformation mithilfe eines Modells zum automatisierbaren Design von Systemintegrationsarchitekturen.

Der Autor verfolgte das Promotionsvorhaben im Zeitraum von 2019 bis 2023 berufsbegleitend zu seiner Tätigkeit als leitender wissenschaftlicher Mitarbeiter in der industriellen pharmazeutischen Forschung und Entwicklung. Dabei adressiert das Forschungsvorhaben mit wissenschaftlichen Methoden wirtschaftliche Notwendigkeiten im Kontext der digitalen Transformation und Modernisierung historischer Systemlandschaften. Dementsprechend ist diese Arbeit in die Profession der Wirtschaftsinformatik einzuordnen, welche Forschung zum Einsatz von Informationstechnologie im wirtschaftlichen Umfeld umfasst.

Abbildung 1 zeigt schematisch den Aufbau dieser Arbeit. Die Arbeit besteht im Kern aus drei aufeinander aufbauenden Teilen Analyse, Synthese und Validierung, welche jeweils in verschiedene Teilbereiche untergliedert sind.

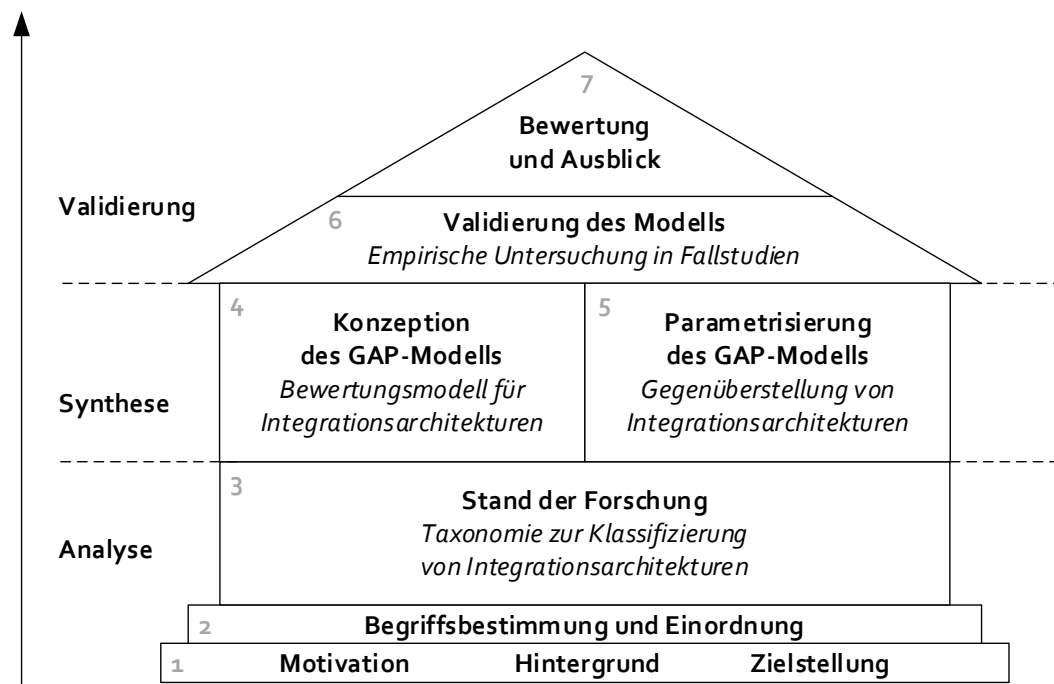


Abbildung 1: Schematische Darstellung des Aufbaus dieser Arbeit

Anmerkung: Bei allen Abbildungen handelt es sich um eigene Darstellungen, wenn nicht explizit im Abbildungstitel auf eine andere Urheberschaft verwiesen wird. Aus diesem Grund verzichtet der Autor in dieser Arbeit auf die dedizierte Kennzeichnung von Abbildungen als „eigene Darstellung“.

Analyse: Anforderungsanalyse anhand analytischer Literaturrecherche und eigenerer vorhergehender Forschungsergebnisse

Der Analyseteil umfasst Kapitel 1, Kapitel 2 und Kapitel 3 und bildet die wissenschaftliche Grundlage dieser Arbeit.

Es erfolgt zunächst eine Bestandsaufnahme, welche auf Literaturarbeit und publizierten Forschungsergebnissen fußt und in der Betrachtung von modellbasierten Ansätzen für Integrationsarchitektur mündet. Der Autor nutzt dabei zur weiteren Vertiefung auch explizit veröffentlichte Forschungsergebnisse, an denen er selbst im Vorfeld dieser Arbeit beteiligt war.

Hierauf aufbauend analysiert der Autor existierende Ansätze zur Klassifizierung von Integrationsarchitekturen und definiert eine Taxonomie, welche Angelpunkt für alle weiteren Überlegungen in dieser Arbeit ist. Ausgehend von einer ausführlichen Literaturrecherche erarbeitet der Autor eine Möglichkeit zur Identifikation von Designartefakten und Segmentierung auf verschiedenen Granularitätsebenen anhand von strukturierten Merkmalen. Im Ergebnis lassen sich unter Anwendung dieser Taxonomie Integrationsarchitekturen anhand ihrer Merkmale archetypisch beschreiben und kategorisieren.

Synthese: Deduktive Konzeption eines Bewertungsmodells

Im Syntheseteil wird die Deduktion eines modellbasierten Ansatzes zur Bewertung von Integrationsarchitekturen beschrieben. Der Forschungsvorgang gliedert sich dabei in die Konzeption des GAP-Modells, sowie die Parametrisierung des GAP-Modells.

Das GAP-Modell als vom Autor konzipiertes Bewertungsmodell für Integrationsarchitekturen bildet den Kernbeitrag dieser Arbeit zum wissenschaftlichen Diskurs. Die zugrundeliegenden Bewertungskriterien sind dabei insbesondere vom BSI-Standard 200 zum IT-Grundschutz, sowie aus der ISO 27000 abgeleitet und bilden die Dimensionen des Bewertungsmodells. Zusätzlich beschreibt der Autor das Vorgehen sowie die holistische Einordnung in ein übergeordnetes Rahmenwerk.

Zur Parametrisierung des GAP-Modells nutzt der Autor die erarbeitete Taxonomie und führt anhand der vorher definierten Bewertungskriterien eine vergleichende Bewertung der sich daraus ergebenden Klassen von Designartefakten von Integrationstopologien, -paradigmen und -methoden durch. Der Autor setzt damit die initialen Messparameter für die spätere Anwendung des GAP-Modells fest und bereitet dieses für die Validierung vor. Die letztendliche Bewertung ist dabei als Hypothese zu verstehen, welche der initialen Parametrisierung des Bewertungsmodells dient und im Rahmen der empirischen Untersuchung anhand von Fallstudien validiert wird.

Validierung: Empirische Untersuchung anhand von Fallstudien

Die Validierung des GAP-Modells erfolgt anhand von drei Fallstudien aus dem beruflichen Kontext des Autors. Dabei erfolgt die Plausibilisierung der Hypothese durch einen qualitativen Vergleich der Ergebnisse des Bewertungsmodells mit Stärken und Schwächen der tatsächlich implementierter Integrationen.

Im Rahmen der Fallstudien erfolgt qualitative Auswertung von identifizierten Architekturmustern und Beobachtungen. Ergänzt werden diese durch Ergebnisse eines Application Assessments in Anlehnung an TOGAF. Dieses Application Assessment erfolgt im Rahmen der unternehmensweiten IT-Architektur-Governance und bewertet im Rahmen des kontinuierlichen Lebenszyklusmanagementprozesses jede Unternehmensapplikation in verschiedenen Dimensionen. Jede Fallstudie betrachtet dabei mehrere verschiedene Implementierungen, so dass die Ergebnisse des GAP-Modells über einen direkten Vergleich dieser Implementierungen zusätzlich plausibilisiert werden können. Da das Application Assessment in seinem Fokus weit über die Systemintegration hinausgeht und explizit die Systeme und nicht die Integrationen zwischen diesen

bewertet, ist eine direkte Übersetzung der vorliegenden Bewertungsergebnisse in die Ergebnisse des zu validierenden Modells nicht ohne weiteres möglich. In Kombination mit der Beschreibung der tatsächlichen Implementierung lassen sich jedoch plausible Zusammenhänge bilden, welche zur Validierung des konzeptionalisierten Modells geeignet sind. In diesem Zusammenhang ist hervorzuheben, dass der Autor keinen Einfluss auf die Ergebnisse dieses Application Assessments hatte, jedoch aus seiner beruflichen Praxis das Kontextwissen zur Auswertung der Bewertungsergebnisse verfügt. Abschließend werden die Ergebnisse der Fallstudien zusammengefasst und im Kontext der Forschungsfragen ausgewertet. Darauf aufbauend begibt sich in eine Retrospektive und bewertet die Forschungsergebnisse, welche durch einen Ausblick ergänzt wird.

Der Autor geht zum Ende auf mögliche Verbesserungen und Weiterentwicklungen des beschriebenen Modells ein und betrachtet zudem weitere Möglichkeiten, wie die digitale Transformation unter den Vorzeichen beschränkter Verfügbarkeit qualifizierter Fachkräfte gelingen kann. So werden konzeptionelle Verbesserungsmöglichkeiten genauso diskutiert, wie auch aktuelle Möglichkeiten zur Automatisierung der Kategorisierung und Bewertung mithilfe künstlicher Intelligenz, sowie die kontinuierliche Verbesserung der Modellparameter durch selbstlernende Algorithmen. Abschließend wendet der Autor seinen Blick zurück auf die anfangs beschriebene Motivation und öffnet eine Perspektive auf Möglichkeiten zur Beschleunigung der digitalen Transformation. Im Ausblick finden sich Überlegungen zur Flexibilisierung der Implementierung von Systemen und ihren Integrationen mit Domänenstandardmodelle und beschreibt Ansätze zur Demokratisierung der Softwareentwicklung als weitere Möglichkeit dem Mangel an informationstechnisch qualifizierten Fachkräften zu begegnen.

2. Begriffsbestimmung und Einordnung

Dieses Kapitel umfasst eine Bestimmung und Einordnung verschiedener im Kontext dieser Arbeit genutzten Begriffe und bildet die Grundlage für das weitere wissenschaftliche Vorgehen.

Ausgehend von einer näheren Beleuchtung der Digitalisierung, sowie der API-basierten Systemintegration erfolgt eine Einordnung in das TOGAF Architekturrahmenwerk, sowie die Betrachtung von Möglichkeiten zur modellbasierten Optimierung von Integrationsarchitekturen. Dabei werden die unter „Modell zur Optimierung der Systemintegration“ zusammengefassten Forschungsfragen FF1.1 (*Wie kann Systemintegration modellgetrieben verbessert werden?*), FF1.2 (*Wie lassen sich Integrationsarchitekturen modellieren?*), sowie FF1.3 (*Wie lässt sich der Transformationsprozess operationalisieren?*) beantwortet.

2.1. Digitalisierung

Das Bundesministerium für Wirtschaft und Klimaschutz definiert Digitalisierung als „Verwendung von Daten und algorithmischen Systemen für neue und verbesserte Prozesse, Produkte und Geschäftsmodelle“ [2] Der Branchenverband Bitkom beschreibt Digitalisierung deckungsgleich als den Einsatz digitaler Technologien zur Steigerung von Effizienz und Produktivität. [14]

Grundlage für die erfolgreiche Digitalisierung ist eine stimmige Technologiebasis, sowie die entsprechenden betrieblichen und organisatorischen Rahmenbedingungen. Digitalisierung bedeutet dabei, dass der Einsatz von Informations- und Kommunikationstechnologie nicht isoliert, sondern immer im betrieblichen Kontext mit dem klaren Ziel der Wertschöpfungssteigerung betrachtet wird. [14]

Der Reifegrad Digitaler Prozesse ist abhängig von einer Vielzahl von Faktoren. Das Reifegradmodell Digitale Prozesse 2.0 definiert die Dimensionen *Technologie*, *Prozessqualität*, *Prozessdaten*, *Kundinnen und Kunden*, sowie *Skills und Kultur*. Jede Dimension wird dabei durch drei Kriterien mit jeweils zwei Fragen bewertet. Eine detaillierte Darstellung des Reifegradmodells findet sich in Anhang 1. [15]

Im Kontext der Wirtschaftsinformatik ist besonders die Dimension der *Technologie* interessant. Diese Dimension betrachtet die technologische Basis und das technologische Umfeld von Geschäftsprozessen und ist in die Kriterien *Technologiebasis*, *Tools im Prozess* und *Systemintegration* unterteilt. Im Rahmen dieser Arbeit wird auf das Kriterium *Systemintegration* fokussiert. In einer guten Integrationslandschaft sind alle im Prozess verwendeten Software-Lösungen vollständig miteinander integriert, wobei der Prozess vollständig ohne Systembrüche abläuft. [15]

2.1.1. Digitale Transformation

Die digitale Transformation beschreibt die Umwälzung, welche mit der Digitalisierung einhergeht. Das Institut der Deutschen Wirtschaft beschreibt diese Entwicklung im wirtschaftlichen Kontext unter dem Schlagwort Industrie 4.0. Durch die Digitalisierung von Geschäftsprozessen eröffnen sich insbesondere mit der Vernetzung, Virtualisierung und intelligenten Datenanalyse völlig neue Wertschöpfungsmöglichkeiten. [23, Seite 50]

Die Art und Weise wie Menschen und miteinander interagieren wird grundlegend revolutioniert. Mit dem Generationenwechsel und den neuen digitalen Möglichkeiten ergeben sich neue Denkmuster und Anforderungen an Geschäftsprozesse. Es ist nun möglich komplexe Berechnungen und Interaktionen in Echtzeit durchzuführen. Somit ergibt sich der Anspruch an eine optimierte Interaktion ohne unnötige Latenzen in Form von Echtzeit-Prozessen. Zudem können Prozesse schlank und effizient gestaltet werden, wenn alle nötigen Informationen vorausschauend aus

den verfügbaren Quellen integriert und vorbereitet worden sind. Prozesse sollten also mit Fokus auf eine gute Nutzbarkeit so gestaltet werden, dass sich der Nutzer auf die wertschöpfenden Aktivitäten im Prozess konzentriert und alle anderen Schritte automatisiert werden. [24, Seite 9–15]

Voraussetzung für den automatisierten Umgang mit Informationen und somit Grundlage für die digitale Transformation ist eine leistungsfähige digitale Infrastruktur. [23, Seite 51] Bestehende Integrationsarchitekturen müssen daher so angepasst werden, dass sie den erhöhten Ansprüchen vernetzter und hochdigitalisierter Geschäftsprozesse gerecht werden können. Dies wird auch durch das Institut der deutschen Wirtschaft im Lagebild zur Digitalisierung in Deutschland bestätigt. Sowohl die Digitalisierung von Prozessen als auch die digitale Vernetzung und Integration von Unternehmen entwickelten sich im Betrachtungszeitraum von 2020 bis 2022 am dynamischsten. [4, Seite 7–10]

2.1.2. Systemintegration

Systemintegration beschreibt die zweckdienliche Kombination von verschiedenen Komponenten, Elementen, und Interaktionen mithilfe von Schnittstellen. [25, Seite 90–92] Im Kontext der Wirtschaftsinformatik bedeutet Systemintegration die Verknüpfung unterschiedlicher Informationssysteme zur Realisierung eines integrierten Geschäftsprozesses. [26] Informationssysteme sind dabei im weitesten Sinne als gekapselte Softwareapplikationen zu verstehen, die bestimmte Informationen im Rahmen eines Prozesses erfassen, verarbeiten und über Schnittstellen zur Verfügung stellen. [10, Seite 3]

Im Reifegradmodell Digitale Prozesse 2.0 der Bitkom ist Systemintegration eines von drei Kriterien zur Bestimmung der technologischen Dimension. Gute Systemintegration zeichnet aus, dass alle in einem Prozess verwendeten Software-Lösungen vollständig miteinander integriert sind und keine unnötigen Systembrüche eingegangen werden. [15]

Ein Systembruch liegt dann vor, wenn bei der Übertragung von Informationen das Medium gewechselt wird. Hierdurch wird die Informationsverarbeitung gebremst und es besteht zusätzlich die Gefahr, dass Informationen verfälscht werden. [27] Daher sollten möglichst alle Informationen entlang der gesamten Prozesskette nach der erstmaligen Erfassung ausschließlich digital über Schnittstellen zwischen den Systemen ausgetauscht werden.

Der Umfang der Integration von Systemen kann variieren. Integrationen auf technischer Ebene werden als Middleware-Integration bezeichnet und beschränken sich auf den reinen Austausch von Daten, ohne dass die Systeme einer gemeinsamen Architektur folgen. Ein Beispiel hierfür ist das Kopieren von Daten aus einer Datenbank in eine andere. Bei Enterprise Application Integration wird zusätzlich Funktionalität integriert, so dass beispielsweise Datenmodelle und Workflows aufeinander abgestimmt werden müssen. Zusätzlich ist es auch möglich, dass die Integration auf der höheren Ebene der Business Architecture stattfindet und beispielsweise organisationsübergreifende Prozesse umgesetzt werden und Domänenstandardmodelle und Masterdaten zum Einsatz kommen. [26]

2.1.3. Konnektivitätsökosystem

Der Begriff des Konnektivitätsökosystems beschreibt die Gesamtheit aller verbundenen Systeme, welche über APIs Daten und Funktionalität miteinander austauschen. [3] Das IEEE beschreibt dies als gigantisches zusammenhängendes Netzwerk von miteinander verbundenen Komponenten, welche in Ihrer Gesamtheit zu einer einzigen programmierbaren Instanz verschmelzen. [28, Seite 64] Die Kommunikation und Integration erfolgt dabei im Sinn der

Plattformökonomie nicht ausschließlich innerhalb der Unternehmensgrenzen, sondern schließt auch die Integration externer Komponenten mit ein. [29]

Alle Akteure im Konnektivitätsökosystem haben Zugriff auf maschinenlesbare und standardisierte Informationen, die mit Ihnen im Ökosystem geteilt werden. Wird in einem Konnektivitätsökosystem auf Zugriffsbeschränkungen verzichtet, so spricht man von Open Data. Unter diesem Gesichtspunkt werden beispielsweise offene Daten der Behörden der unmittelbaren Bundesverwaltung frei verfügbar zum Datenabruf über öffentlich zugängliche Netze bereitgestellt. [30]

2.1.4. Akteure im Konnektivitätsökosystem

Der Autor orientiert sich bei der Kategorisierung der Akteure im Konnektivitätsökosystem am NIST Big Data Interoperability Framework (NBDIF) des National Institute of Standards and Technology (NIST). NBDIF definiert für Big Data Systeme folgende Stakeholder: Data Provider, Data Consumer, System Orchestrator, Big Data Application Provider und Big Data Framework Provider. [31, Seite 10–11]

Im Kontext von Systemintegration sind die Rollen Data Provider und Data Consumer relevant, da diese die Informationen in das Ökosystem einbringen bzw. verarbeiten. Die entsprechende Kategorisierung nach der Rolle beim Informationsaustausch ist in Folgendem beschrieben und in Abbildung 2 dargestellt. Die Bezeichnungen werden dabei auf Provider und Consumer vereinfacht und um die kombinierte Rolle des Prosumers ergänzt. Die Klassifizierung erfolgt dabei isoliert für die jeweils geteilten Informationsobjekte.

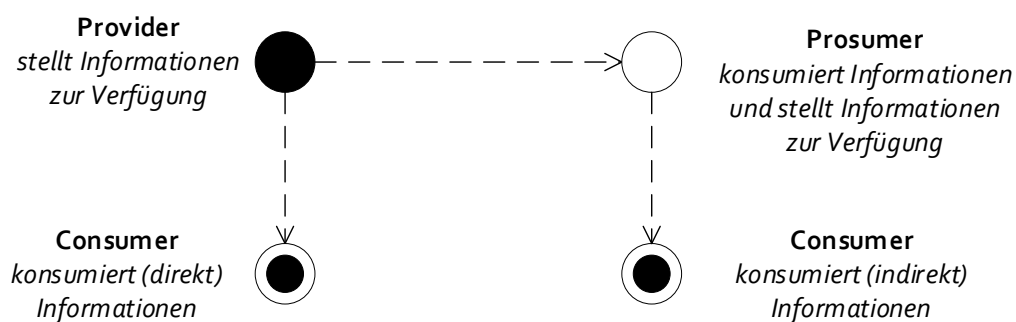


Abbildung 2: Kategorisierung von Akteuren im Konnektivitätsökosystem

Provider

Provider stellen dem Ökosystem neue Informationen bereit. Diese Systeme erfassen, sammeln und speichern die Daten und stellen diese innerhalb des Ökosystems bereit. [31, Seite 14–15]

Provider sind der Startpunkt im Pfad und gehen dementsprechend ausgehende Verbindungen mit anderen Akteuren ein. Provider sind somit die Quellsysteme für die mit ihnen verbundenen Systeme.

Eine Änderung an diesen Systemen hat Einfluss auf alle ausgehenden Verbindungen. Ändert sich beispielsweise ein Datenfeld in einem Formular, muss dieser Änderung auch in den verbundenen Systemen begegnet werden.

Ein Beispiel für ein Provider-System ist ein Dateneingabesystem, in welches Nutzer Informationen in Eingabemasken übermitteln können, welche dann als Quelle für alle weiteren Prozesse dienen.

Consumer

Consumer sind das Gegenstück vom Provider und nutzen die vom Provider bereitgestellten Informationen beispielsweise zur Analyse, Visualisierung, oder weiteren Verarbeitung. [31, Seite 16–17]

Consumer sind somit ein Endpunkt im Pfad und haben keine ausgehenden Kanten. Sie sind als Zielsystem abhängig von den eingehenden Integrationen der Quellsysteme. Änderungen an den vorgelagerten Systemen wirken sich direkt auf die Consumer-Systeme aus.

Ein Beispiel für ein Consumer-System ist eine Reporting-Applikation, welche Kennzahlen aus verschiedenen Systemen erhält und diese in einem Dashboard zur Analyse der Geschäftsergebnisse grafisch aufarbeitet.

Prosumer

Prosumer bezeichnet Konsumenten, welche gleichzeitig Produzenten sind. [32] Sie vereinen somit die Eigenschaften beider Klassen. Sie stellen beispielsweise im Ökosystem Informationen zur Verfügung, welche durch die Verarbeitung der Informationen eines Providers entstanden sind. [31, Seite 14]

Provider sind Knoten im Pfad und haben sowohl eingehende als auch ausgehende Kanten. Sie sind somit sowohl Zielsysteme als auch Quellsysteme und können Informationen dabei entweder unverändert weitervermitteln, oder in verarbeiteter Form zur Verfügung stellen. Genau wie bei Provider-System hat eine Änderung Einfluss auf die ausgehenden Verbindungen. Dabei müssen sowohl Anpassungen an der Verarbeitungslogik im Prosumer-System als auch Änderungen am ursprünglichen Provider-System betrachtet werden.

Ein Beispiel für ein Prosumer-System ist ein Shopsystem, welches vom Warenwirtschaftssystem den aktuellen Lagerbestand erhält, diese verarbeitet und dem Dispositionssystem für die Versandvorbereitung zur Verfügung stellt.

2.2. API-basierte Systemintegration

„APIs sind der „Kleber“, der die Elemente einer Systemarchitektur zusammenhält.“

So beschrieben Perry und Wolf in Ihrem Beitrag „Foundations for the Study of Software Architecture“ 1992 die Rolle der Elemente einer Systemlandschaft, welche für die Integration der verschiedenen Systeme genutzt werden. [33]

2.2.1. Application Programming Interface (API)

Das Institute of Electrical and Electronic Engineers (IEEE) definiert den Begriff Application Programming Interface (API) als standardisierte Softwareschnittstelle. Diese erlaubt es Informationssystemen miteinander in Interaktion zu treten und Informationen auszutauschen. [34] APIs sind somit die Grundlage für Systemintegration, da erst über APIs der Informationsaustausch zwischen den Systemen automatisiert werden kann. Der Begriff API steht somit als Bezeichnung für alle digitalen Schnittstellen, über welche sich Systeme miteinander verknüpfen lassen.

APIs können unterschiedlich entworfen und implementiert werden. Hierbei ist explizit hervorzuheben, dass der API-Begriff alle Systemschnittstellen umschließt und nicht auf Web-APIs begrenzt ist. Vielmehr müssen IT-Verantwortliche dafür sorgen, dass die jeweiligen Schnittstellen den aktuellen Marktanforderungen entsprechen. [35]

Mithilfe von APIs als maschinenlesbaren Schnittstellen lassen sich Systemfunktionen abstrahieren und für andere Systeme bereitstellen, so dass Informationen, Daten, Funktionalitäten und Fähigkeiten zwischen verschiedenen Systemen integriert werden können. Die Integration über APIs erfolgt dabei idealerweise standardisiert und ermöglicht eine Modularisierung der Systemlandschaft. Die hierdurch eröffneten Verwertungspotentiale durch digitale Geschäftsmodelle und Wertschöpfungsnetzwerke sind ein klarer Treiber der Digitale Transformation. [35]

2.2.2. API-fizierung

API-fizierung beschreibt den Prozess Fähigkeiten und Informationen über APIs in einem Konnektivitätsökosystem zur Verfügung zu stellen. Daten und Funktionalitäten werden dabei konsequent standardisiert und über eine maschinenlesbare Zugriffsschicht bereitgestellt. Im Zuge der API-fizierung werden somit APIs gezielt aufgebaut, verbessert, und verfügbar gemacht. [3] Dieser „API-first“-Gedanke ist eine der Grundlagen für den Erfolg von Firmen wie Amazon und Google. [36] Diese wurde vom Autor auch durch eine Empirische Untersuchung zum Stand der APIfizierung im Kontext des „Berliner Forum Digitalisierung“ bestätigt. Bei der Befragung gab ein Großteil der Befragten an, dass APIs im Unternehmen bereits eine entscheidende Rolle spielen. APIs werden dabei sowohl für die Verknüpfung unternehmensintern und mit externen Partnern genutzt als auch zur öffentlichen Nutzung freigegeben. [37]

Laut der Erhebung „The State of API“ aus dem Jahr 2019 war der Hauptgrund für die API-fizierung im Unternehmensumfeld eine verbesserte Systemintegration. [38] Relevante Auszüge aus dieser Studie finden sich in Anhang 3.

Mit der API-fizierung werden Systemgrenzen durchlässig und bestehende Fähigkeiten lassen sich zu neuen Wertschöpfungsketten orchestrieren. Zudem wird die Systemlandschaft entkoppelt und einzelne Komponenten können weiterentwickelt und ausgetauscht werden, ohne dass dies die operative Integrität des Gesamtsystems gefährdet. [39] API-fizierung ist auch ein Vehikel zur verbesserten Vernetzung von Applikationen entlang der Wertschöpfungskette und erlaubt zudem die Abstraktion komplexer Abläufe und Logiken hinter einfachen API-Fassaden. Die API-

fizierung ist somit ein Werkzeug, um monolithische Strukturen aufzubrechen und die Systemlandschaft im Sinne eines dynamischen Ökosystems flexibel miteinander interagierender schlanker Produkte zu modernisieren. [22]

Die API-fizierung von Altanwendungen ist eine besondere Herausforderung, da alte Systeme oftmals nicht den aktuellen technischen und konzeptionellen Anforderungen gerecht werden. [35] Dabei ist es nicht immer möglich eine bestehende Lösung durch eine neue zu ersetzen, ohne hohe Kosten, oder eine Gefährdung der operativen Integrität in Kauf zu nehmen. Vielmehr bietet die nachträgliche API-fizierung von Altanwendungen die Möglichkeit, gereifte Funktionalität über eine standardisierte Schnittstelle im Konnektivitätsökosysteme verfügbar zu machen. [40] Das Thema der Massiven API-fizierung von Legacy Applikationen wurde auch als Thema einer Expertendiskussion im Rahmen des 4. Workshop Evaluation of Service-APIs – ESAPI 2020 unter dem Motto „APIs als Klebstoff einer allumfassenden Digitalisierung“ diskutiert. Dabei wurde erörtert, dass für eine erfolgreiche API-fizierung neben den technischen Aspekten auch kulturelle Aspekte eine Rolle spielen. Eine standardkonforme und gut dokumentierte Implementierung von Schnittstellen erfordert eine offene und kollaborative Kultur. Dabei ist die Akzeptanz von API-fizierung auch abhängig von den tatsächlichen Vorteilen für alle Teilnehmer am Konnektivitätsökosystem. Ein Schaubild zu dieser Diskussion findet sich in Anhang 6 auf Seite 153.

Zusammenfassend ist hervorzuheben, dass API-fizierung den Wandel von isolierten Wertschöpfungsketten hin zu einer Plattformökonomie ermöglicht. [3] Zudem bietet sich ein mächtiges Werkzeug zur Verbesserung von Datenverfügbarkeit und der Konnektivität. [41] Voraussetzung hierfür sind eine gute technische Infrastruktur und gut konventionalisierte APIs. Die API-fizierung ermöglicht somit eine hohe Wertschöpfung in der API-Economy und ist daher ein klarer Treiber der Digitalisierung. [35]

2.2.3. API-Management

API-Management beschreibt die Gesamtheit aller Maßnahmen, die zur Verwaltung und Gestaltung von APIs ergriffen werden. Dies umfasst insbesondere eine Katalogisierung von Schnittstellen, Definition und Umsetzung von technischen und betrieblichen Standards, sowie eine Übersicht über die Nutzung verschiedener APIs. [42]

API-Management dient als Werkzeug zur kontinuierlichen Optimierung der Systemintegration. So ist das in Abbildung 3 dargestellte Full-Lifecycle-API-Management-Modell ein konkreter Ansatz zur zyklischen Architekturgestaltung, welcher vom Autor bereits in anderen Publikationen beschrieben wurde. [43, Seite 37–44]

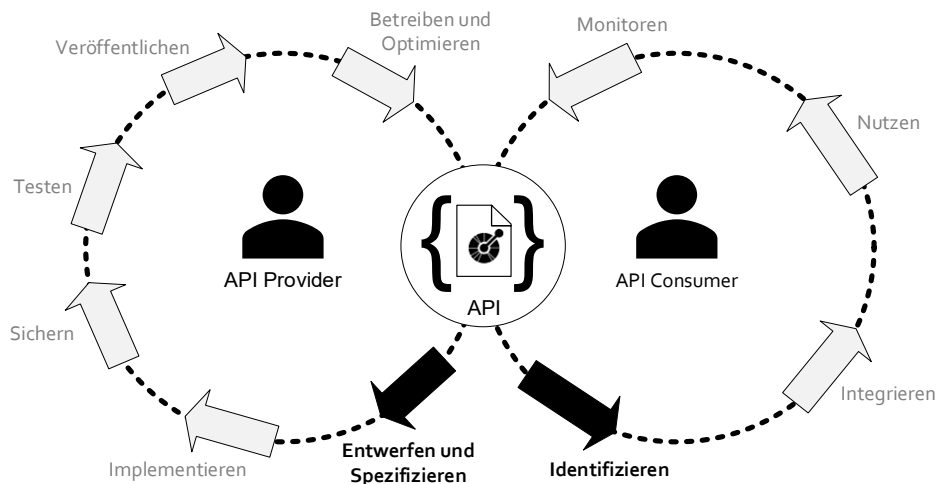


Abbildung 3: Full-Lifecycle-API-Management-Modell
(Quelle: Schmietendorf et al. [43, Seite 39] in Anlehnung an Chiu [44])

Im Full-Lifecycle-API-Management-Modell durchläuft eine API mehrere miteinander verknüpfte Lebenszyklusphasen, welche entweder vom API Provider, oder API Consumer getrieben werden. Dabei beginnt der Lebenszyklus einer API mit der Integrationsstrategie und dem entsprechenden Design der jeweiligen Schnittstelle. Anschließend wird diese vom Provider zur Verfügung gestellt und vom Consumer genutzt. [44] Im Kontext dieser Arbeit sind insbesondere die Phasen „Identifizieren“, sowie „Entwerfen und Spezifizieren“ relevant.

Die Identifikationsphase auf Seite des API-Consumers beschreibt, wie potenzielle Nutzer bestehender APIs diese identifizieren und auswählen. [43, Seite 41] So kann eine Architekturentscheidung für die Integration bestehender Systeme maßgeblich von den bereits zur Verfügung stehenden Schnittstellen und Technologien abhängen. Ein bereits systemseitig zur Verfügung gestellte REST-API wäre so beispielsweise einem neu zu Implementierenden Eventstreaming vorzuziehen. Dieser Aspekt wird in den Abschnitten 3.4 und 5.3 unter dem Terminus „native API“ und „Wrapper/Mediator“ näher beleuchtet.

Die Entwurfs- und Spezifikationsphase auf Seite des API-Providers beschreibt den implementierungsorientierten Entwurf der API [43, Seite 39]. Entsprechend der optimalen Zielarchitektur kann dieser Entwurf gezielt gesteuert werden, so dass die implementierte API zur definierten Zielarchitektur passt. Das API-Lifecycle-Management steht somit in enger Wechselwirkung mit dem Management der Integrationsarchitektur und bedarf somit einer engen Abstimmung.

Zusammenfassend lässt sich API-Management als generelles Konzept zur professionellen Beherrschung des Konnektivitätsökosystems beschrieben. Anhand der Entwicklung in Bezug auf API-Management lässt sich die fortschreitende Professionalisierung von Systemintegration darstellen. In der Vergangenheit wurde das Thema API-Management hauptsächlich aus einer rein

technischen Perspektive eingeordnet und war synonym mit einem Integrationsgateway. Innerhalb der letzten Jahre hat sich das Bild hin zu einem ganzheitlichen Ansatz gewandelt, welcher den gesamten API-Lebenszyklus abbildet. [44] Diese Entwicklung wird auch durch entsprechende Analysen von Gartner bestätigt, welche sich seit 2016 jährlich mit dem Thema API-Management beschäftigen. [45]

2.2.4. API-Management-Plattform

Das Konzept von API-Management Bedarf für die Umsetzung im großen Maßstab neben dem entsprechenden Fachwissen auch professionelle Werkzeuge und Infrastruktur. Eine API-Management-Plattform stellt diese Infrastruktur und Werkzeuge zur Verfügung. Eine typische API-Management-Plattform besteht dabei aus verschiedenen Komponenten, die eine sichere Kommunikation zwischen Systemen ermöglichen und zur Verwaltung der Schnittstellen geeignet sind. [46, Seite 23–34]

Abbildung 4 zeigt schematisch mögliche Komponenten einer API-Management Plattform bestehend aus einem API-Gateway, einem API-Katalog, sowie verschiedenen Modulen zur Realisierung der Systemintegration.

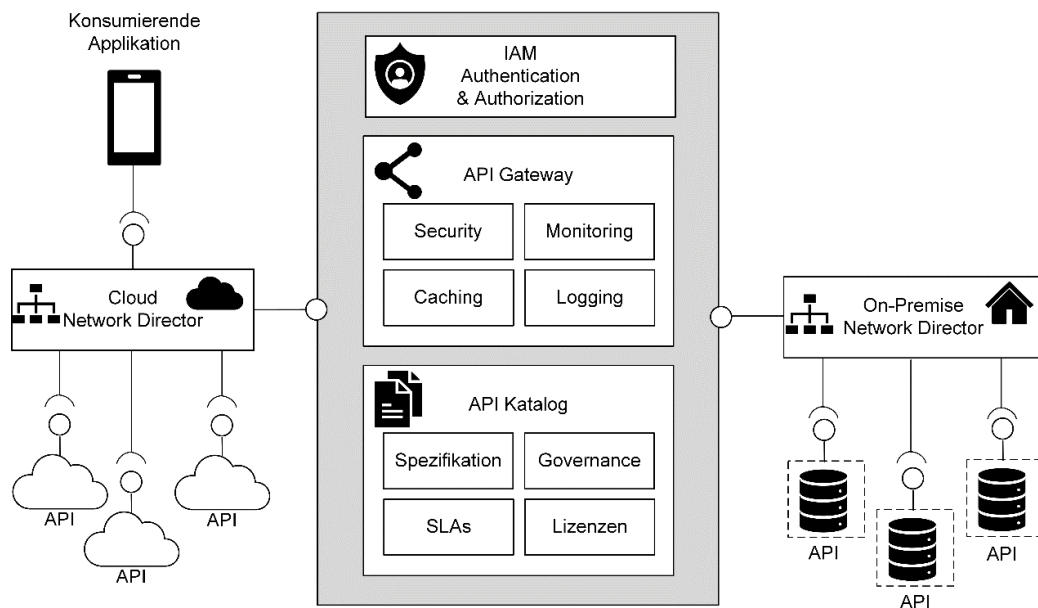


Abbildung 4: Schematische Darstellung einer API-Management-Plattform
(Quelle: Schmietendorf et al. [43])

2.3. IT-Architektur

Der Begriff IT-Architektur bezeichnet die fundamentalen Konzepte, Prinzipien und Eigenschaften von Informationssystemen und ihrer Umgebung. Dabei werden insbesondere die Beziehungen und Interaktionen zwischen einzelnen Komponenten definiert und Module nach einem bestimmten Architekturstil mit festen Regeln und Mustern ausgestaltet. [47, Seite 4]

IT-Architektur ist inhärenter Teil des IT-Managements und ist bestimmend im Gestaltungsprozess und Lebenszyklus von IT-Landschaften. Dies ist beispielsweise im The Open Group Architecture Framework beschrieben. TOGAF ist das führende Framework für Enterprise Architektur [5] und wird daher in Folgendem als Referenzstandard für IT-Architektur genutzt. Aus den folgenden Betrachtungen ergeben sich Antworten auf die Forschungsfrage FF 1.1 (*Wie kann Systemintegration modellgetrieben verbessert werden?*)

Es handelt sich bei TOGAF um eine Sammlung von Richtlinien und Empfehlungen, welche in mehreren eng miteinander verknüpften Dokumenten beschrieben sind. Ziel ist die Etablierung eines gemeinsamen Standards, eine gemeinsame Herangehensweise und der Austausch von Best-Practices. [6, Seite VII]

Wie in Abbildung 5 dargestellt, definiert das TOGAF Content Framework Architekturmanagement als Zusammenspiel verschiedener Aufgabenbereiche.

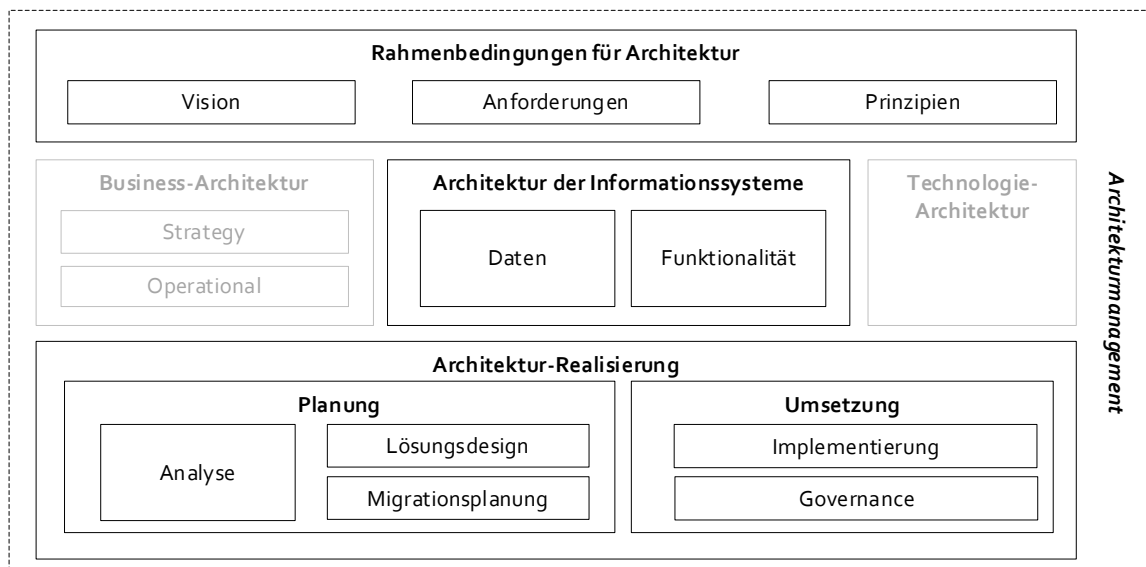
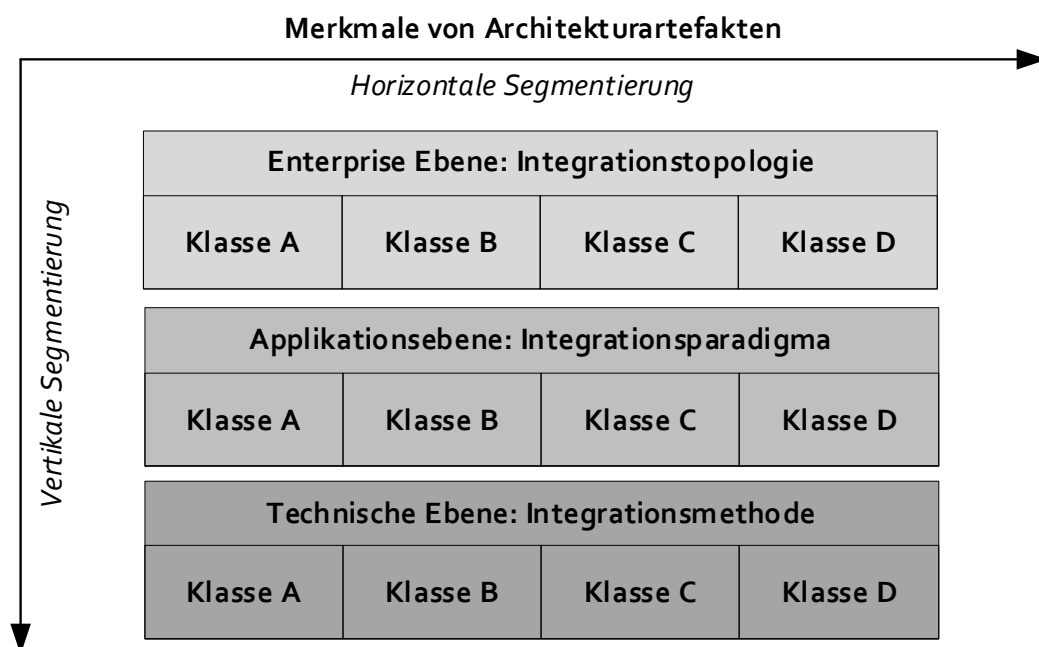


Abbildung 5: TOGAF Content Framework
(Quelle: Eigene Darstellung in Anlehnung an The Open Group [48, Seite 5])

IT-Architektur, bzw. die Architektur der Informationssysteme fokussiert auf die in der Organisation verfügbaren Daten und Funktionalitäten. Die Rahmenbedingungen für die Architektur ergeben sich aus der Architekturvision, den spezifischen Anforderungen, sowie den zugrundeliegenden Prinzipien. Die Gestaltung der IT-Architektur erfolgt in der Architektur-Realisierung durch Planung und Umsetzung von Architekturentscheidungen. Basierend auf einer Zustandsanalyse und Bewertung jeweiligen Chancen und Risiken entsteht im Kontext der IT-Architektur ein individuelles Lösungsdesign, sowie eine Migrationsplanung, welche den Plan für die Transformation beschreibt. Diese Architekturentscheidungen werden dann über anwendungsspezifische Implementierung und allgemeine Governance umgesetzt. [48, Seite 5–6]

Die Architektur der Informationssysteme kann auf verschiedenen Granularitätsebenen mit unterschiedlicher Detailtiefe betrachtet werden. Eine tiefere Detailtiefe ermöglicht und erfordert eine detailliertere Modellierung der jeweiligen Designartefakte [9, Seite 23–25]. Im Kontext dieser Arbeit wird auf die Architektur von Informationssystemen und insbesondere auf die Ausgestaltung von Integrationslandschaften und den Schnittstellen zwischen den Systemen (API) fokussiert. APIs stehen dabei in enger Wechselwirkung mit der globalen Architektur der Informationssysteme. Zum einen sind bestehende APIs Einflussfaktor für die Architekturentscheidungen, zum anderen spiegeln sich Architekturentscheidungen in der Implementierung von Schnittstellen und Systemintegrationen wieder. Wie in Abbildung 6 dargestellt, lassen sich hierfür die drei Granularitätsebenen **Enterprise-Ebene** (Integrationstopologie), **Applikationsebene** (Integrationsparadigma) und **Technische Ebene** (Integrationsmethode) definieren, wobei mit tieferer Detailtiefe eine Segmentierung in Teilbereiche stattfindet. [9, Seite 25]



*Abbildung 6: Granularitätsebenen von Informationssintegration
(Quelle: Eigene Darstellung in Anlehnung an The Open Group [9, Seite 25])*

Der Detailgrad der Modellierung kann bei Bedarf weiter verfeinert werden. So könnte beispielsweise das technische Systemdesign als weitere Granularitätsebenen abgebildet werden. Für die Bewertung der Integrationsarchitekturen ist diese Detailschärfe jedoch irrelevant und wird daher im Kontext dieser Arbeit nicht weiterverfolgt.

2.3.1. Enterprise-Ebene zur Einordnung von Integrationslandschaften

Die Enterprise-Ebene enthält die Gesamtheit aller IT-Systeme und -Anwendungen innerhalb einer Organisation und stellt damit den größten Detailgrad der Betrachtung dar. Eine Organisation, bzw. ein „Enterprise“ ist dabei als ein abgrenzbarer Zusammenschluss verschiedener Organisationseinheiten mit einem gemeinsamen Zweck definiert. Dies kann beispielsweise ein Unternehmen, eine Konzerndivision, ein Ministerium, eine Hochschule, oder ein Konsortium sein. [6, Seite 2]

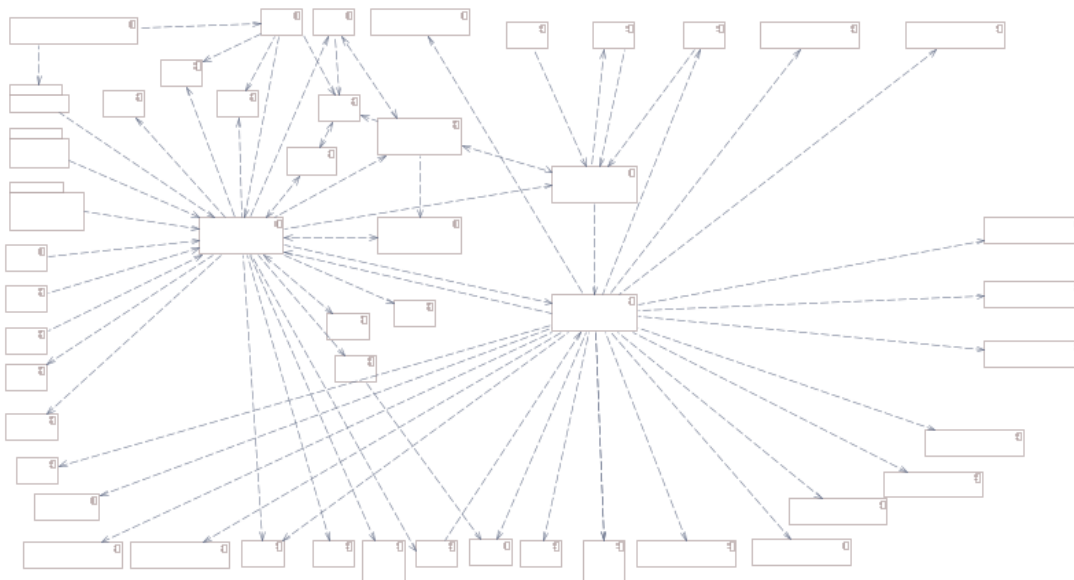


Abbildung 7: Beispielhafte Darstellung einer Integrationslandschaft auf Ebene der Enterprise Architecture

Die Enterprise-Ebene betrachtet die Integrationsarchitektur als strategisches Element und bildet die Interaktion der einzelnen Architektursegmente im Gesamtkontext ab. [9, Seite 24–25] Abbildung 7 stellt exemplarisch eine solche Architektur auf größter Detailebene dar und zeigt, welche Systeme miteinander verbunden sind. Dabei wird die konzeptionelle Gestaltung der Interaktionen und des Informationsaustausches zwischen miteinander interagierenden Systemen betrachtet. Auf dieser Detailebene werden die Informationssysteme als gekapselte Einheit begriffen und es wird nur die Verbindung zwischen den verschiedenen Systemen betrachtet ohne auf systeminterne Komponenten, oder technische Details des Informationsaustausches einzugehen. Es ist zu beachten, dass Integration und Interaktion auch über die Organisationsgrenzen hinaus möglich sind.

Aufbauend auf diesem Verständnis geht der Autor in Abschnitt 3.2 näher auf die Möglichkeiten zur Segmentierung von Integrationslandschaften auf Enterprise-Ebene ein und beschreibt eine Kategorisierung in verschiedene Integrationstopologien.

2.3.2. Applikationsebene zur Einordnung von Integrationsmustern

Die Applikationsebene beschreibt die Architektur mit Fokus auf einzelne Applikationen innerhalb der Systemlandschaft und definiert somit ein ausgewähltes Segment der Enterprise-Architektur in einem höheren Detailgrad. Die Komplexität der einzelnen Applikationen und ihrer APIs lässt sich damit kapseln und für den jeweiligen Detailgrad abstrahieren. Dabei ist hervorzuheben, dass eine Applikation abhängig von der Definition der Applikationsgrenzen aus mehreren technischen Systemen bestehen kann. Eine Applikation kann beispielsweise aus einer Kombination von wiederverwendbaren Modulen bestehen, die sich auch in anderen Detaildarstellungen auf Applikationsebene wiederfinden. [9, Seite 21] Ein klassisches Beispiel hierfür wäre ein zentrales Data Warehouse, welches von verschiedenen Applikationen genutzt wird und somit auch Teil von verschiedenen Applikationsarchitekturen sein kann.

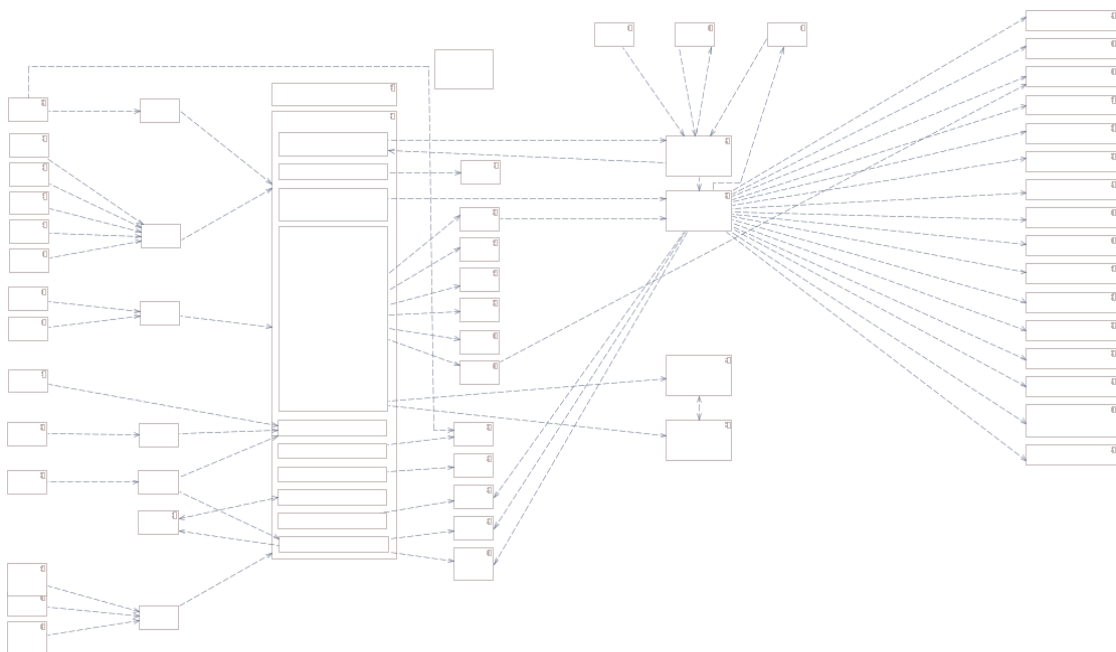


Abbildung 8: Beispielhafte Architekturdarstellung auf Ebene der Application-Architecture

Abbildung 8 zeigt eine beispielhafte Architekturdarstellung auf Ebene der Application-Architecture bestehend aus mehreren technischen Systemen. Auf dieser Ebene sind alle Module dargestellt, die direkt oder über Middleware-Komponenten miteinander verbunden sind. Die Grenzen von Applikationsarchitekturen können dabei zum Beispiel durch eine Gruppierung von Anwendungsfällen in einem bestimmten Fachgebiet definiert sein.

Hierauf aufbauend beschreibt der Autor in Abschnitt 2.3.2 Möglichkeiten zur Segmentierung von Architekturen auf Applikationsebene. Die jeweiligen Integrationsmuster werden dabei dem Integrationsparadigma zugeordnet, welches den jeweiligen Verknüpfungen zugrunde liegt.

2.3.3. Technische Ebene zur Einordnung von Integrationsverfahren

Die Technische Ebene beschreibt die Lösungsarchitektur, bzw. „Solution-Architecture“ und definiert die technische Implementierung einer bestimmten Komponente innerhalb der Systemlandschaft. Anders als die höheren Abstraktionsebenen enthält die Architekturbeschreibung in diesem Detailgrad alle Informationen des Architekturentwurfs, die für die Implementierung relevant sind. Hierbei werden im Sinne von technischen Best Practices explizit Standardkomponenten und wiederverwendbare Module genutzt. [9, Seite 20–25]

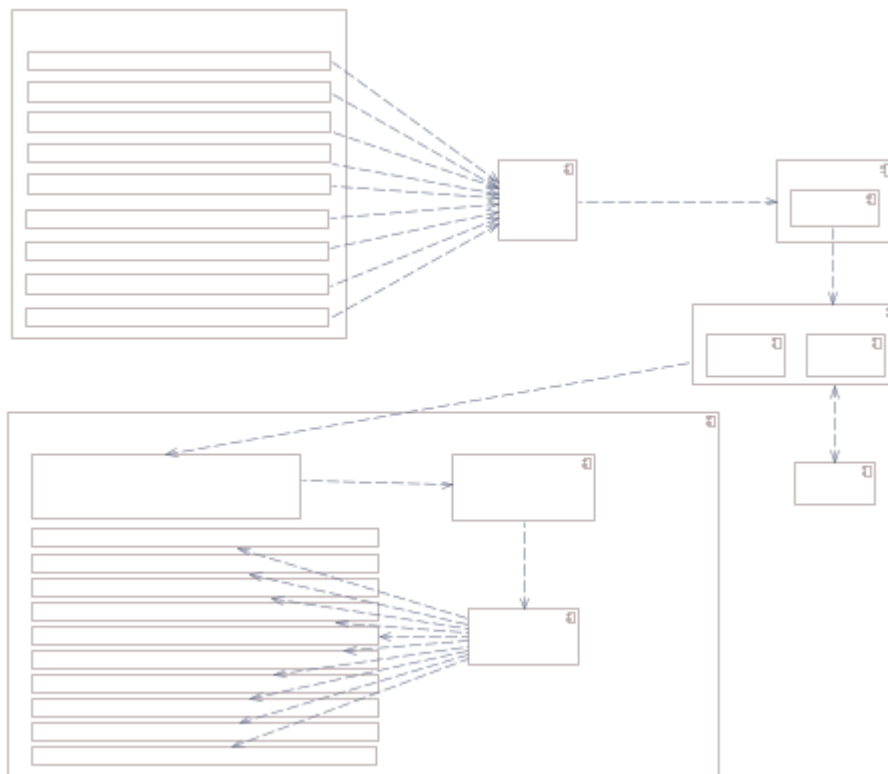


Abbildung 9: Beispielhafte Darstellung einer Lösungsarchitektur auf technischer Ebene

Abbildung 9 zeigt eine beispielhafte Darstellung einer Lösungsarchitektur auf technischer Ebene. Es handelt sich hierbei um die Ausgestaltung einzelner Komponenten, welche in Ihrem Zusammenspiel bereits auf Ebene der Applikationsarchitektur enthalten sind. Die Lösungsarchitektur beschreibt dabei zum einen die Kombination von technischen Bausteinen zu einer Lösung innerhalb einer Applikation und zum anderen die Funktionalität und der einzelnen Module.

Im Abschnitt 3.4 beschreibt der Autor, wie Systemintegrationen auf technischer Ebene segmentiert werden können. Die Klassifizierung erfolgt dabei auf Grundlage des technischen Verfahrens, mit welcher die Integration umgesetzt wurde. Dabei wird jede Integration einer Methode zugeordnet, welche für die Realisierung der Integration genutzt wurde.

2.4. Modellbasierte Ansätze zur Optimierung von Integrationsarchitekturen

2.4.1. Standardisierte Notation der Architektur mit UML

Die Modellierung ist eine unverzichtbare Disziplin der IT Architecture. Eine zweckdienliche Modellierung der Abhängigkeiten bildet die Grundlage, um die Komplexität im Konnektivitätsökosystem zu beherrschen und letztendlich Änderungen an Systemen durchführen zu können, ohne die operative Integrität des Ökosystems zu gefährden. Anhand der modellierten Informationspfade lassen sich Abhängigkeiten in komplexen Ökosystemen abbilden. So kann beispielsweise erfasst werden, welchen Einfluss Änderungen an bestimmten Systemen auf die jeweils verbundenen Systeme hat.

Die Unified Modeling Language (UML) ist der etablierte Standard für die Notation von Systemmodellen und ermöglicht eine übertragbare und maschinenlesbare Beschreibung aller relevanten Komponenten. UML bietet als Werkzeug IT-Architekten, Softwareingenieuren, Softwareentwicklern und anderen Experten die Möglichkeit, Strukturen, Verbindungen und Prozesse in Systemen und Systemlandschaften präzise abzubilden und standardisiert zu modellieren. [49, Seite 43]

Insbesondere in Hinblick auf Möglichkeiten zur Automatisierung wird eine maschinenlesbare Notation benötigt, in der Zusammenhänge und Wechselwirkungen in der Integrationslandschaft klar abgebildet werden können. UML bietet diese Möglichkeiten zur klaren Definition von Modulen, Akteuren, Artefakten und Akteuren im Konnektivitätsökosystem.

In Bezug auf die Forschungsfrage FF 1.2 (*Wie lassen sich Integrationsarchitekturen modellieren?*) ergibt sich die Antwort mit UML als Möglichkeit zur Modellierung von Integrationsarchitekturen auf den in Abschnitt 2.3 beschriebenen Granularitätsebenen der IT-Architektur

Abbildung 10 zeigt die Modellierung von Integrationsarchitekturen auf Enterprise-Ebene. Dabei wird abgebildet, welche Systeme miteinander verbunden sind und welche Informationen sie miteinander teilen.

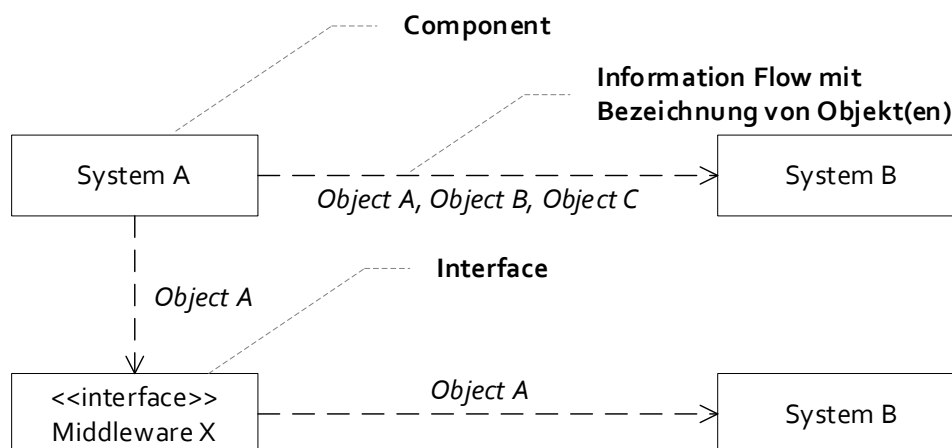


Abbildung 10: UML zur Modellierung von Systemintegration auf Enterprise-Architecture-Ebene

Alle Systeme einer Integrationslandschaft werden als „Component“ modelliert. [49, Seite 208–215] Integrationen zwischen zwei Systemen können als „Interface“ modelliert werden, wenn es

keinen direkten Informationsaustausch zwischen zwei Systemen gibt. [49, Seite 170–172] Für die Modellierung dieser Zusammenhänge können „Information Flows“ genutzt werden. Dabei werden Informationsflüsse zwischen zwei Komponenten als Pfeile, bzw. gerichtete Kanten dargestellt. [49, Seite 669–676]

Im Rahmen dieser Arbeit wird für schematische Darstellungen eine vereinfachte Form der Modellierung genutzt. Um eine klare visuelle Abgrenzung zu realen Modellen zu erreichen, werden Systeme bei konzeptionellen Darstellungen vom UML-Standard abweichend mit Kreisen anstatt mit Rechtecken repräsentiert. Eine Verbindung zwischen zwei Systemen wird analog zum Information Flow als gerichteten Kante abgebildet, bei der jedoch natürlich keine Objekte hinterlegt werden (Information Flows). Die Richtung der Kante stellt dabei den Pfad dar, den eine bestimmte Information nimmt. [50, Seite 1–34]

Wie in Abbildung 11 schematisch dargestellt, lässt sich im Modell für jede Information ein Pfad abbilden. Startpunkt ist dabei das Quellsystem, also jenes System, in welchem eine Information ursprünglich generiert wird. Die Übertragung von Information von einem in ein anderes System ist als gerichtete Kante vom Quell- ins Zielsystem dargestellt. Dabei kann der Informationspfad einen Startknoten, jedoch beliebig viele Mittel- und Endknoten haben. Anhand dieser Modellierung können nun Abhängigkeiten identifiziert, Integrationsstrukturen analysiert und Architekturmuster identifiziert werden.

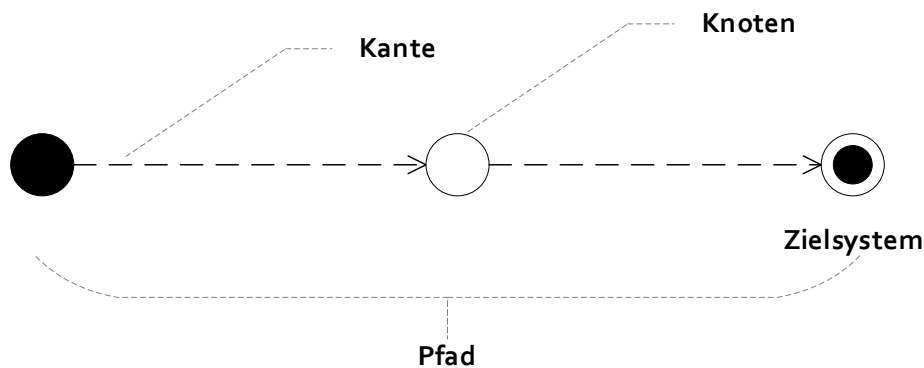


Abbildung 11: Vereinfachte Notation von Integrationspfaden

Die Modellierung auf Applikationsebene und Technischer Ebene erfolgen analog, jedoch in detaillierterer Auflösung und unter Ausnutzung weiterer Modellierungselemente von UML. So können einzelne Components in Pakete zusammengefasst, oder explizit als Interface ausmodelliert werden. Auf Technischer Ebene kann diese Modellierung zudem mit Ablaufdiagrammen ergänzt werden. Diese Art der Modellierung wird vom Autor beispielsweise bei der Darstellung der Anwendungsfälle in der Fallstudie in Kapitel 6 genutzt.

2.4.2. Standardisiertes Management der IT-Architektur mit dem TOGAF Content Framework

Ausgehend von den oben beschriebenen Anforderungen wird ein Werkzeug für das Management von Integrationsarchitekturen benötigt. Das in Abschnitt 2.3 bereits referenzierte TOGAF Content Framework gibt einen Rahmen für das Architektur-Management und definiert verschiedene Aufgabenbereiche der IT-Architektur. Hieraus ergeben sich Möglichkeiten zur Beantwortung der Forschungsfrage FF 1.3 (*Wie lässt sich der Transformationsprozess operationalisieren?*)

Wie in Abbildung 12 dargestellt, umfasst TOGAF eine Sammlung von Ansätzen, anhand derer die Architektur strukturiert entwickelt und beschrieben werden kann. Grundgedanke ist die Anforderungsbasierte Entwicklung und kontinuierliche Gestaltung von Architektur. [51, Seite 4] Grundsätzliche Prinzipien, Architekturvision und individuelle Anforderungen bilden dabei die Rahmenbedingungen für Architektur. [6, Seite 30]

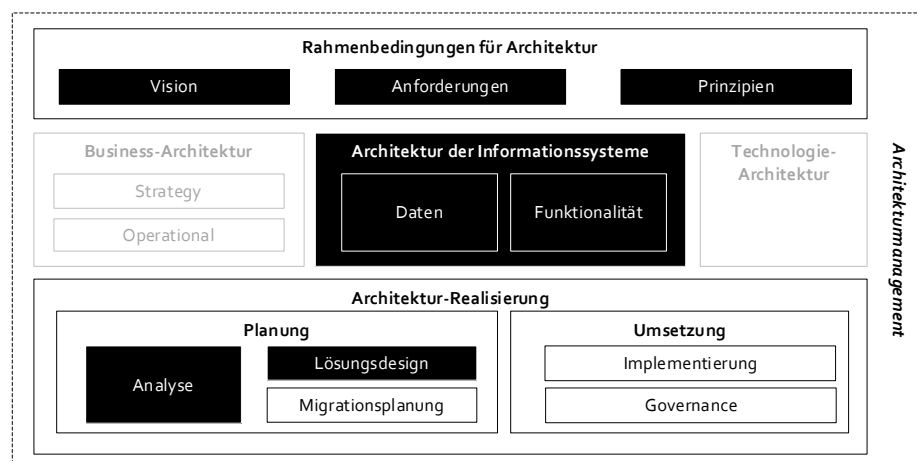


Abbildung 12: Einordnung des GAP-Modells in das TOGAF Content Framework
(Quelle: Eigene Darstellung in Anlehnung an The Open Group [48, Seite 5])

Ausgehend von den Rahmenbedingungen wird die Architektur definiert. Die Bedeutung guter Informations- bzw. Systemintegration wird dabei in TOGAF unter dem Aspekt der „Interoperability“ explizit hervorgehoben. [6, Seite 25]

Bei der Definition der Architektur folgt TOGAF einem ganzheitlichen Ansatz. So ist die IT-Architektur von der Business-Architektur abhängig und mit der Technologie-Architektur eng verbunden. Die Business-Architektur beschreibt insbesondere die Geschäftsziele, sowie die Fähigkeiten und Prozesse einer Organisation. Die Technologie-Architektur umfasst die Beschreibung der technischen Komponenten, welche für die Implementierung der Informationssysteme zum Einsatz kommen. Im Kontext dieser Arbeit fokussiert der Autor auf die Architektur der Informationssysteme als Beschreibung von Daten und Funktionalität innerhalb der Systemlandschaft. [6, Seite 30–31]

Als drittes Element beschreibt TOGAF die Realisierung der Architektur, wobei zwischen Planung und Umsetzung unterschieden wird. Hierbei werden insbesondere zukünftige Lösungsdesigns modelliert und Maßnahmen zur Umsetzung der Architektur abgebildet. [6, Seite 31]

2.4.3. Standardisierte Architekturentwicklung mit der TOGAF Architecture Development Method

TOGAF beschreibt den Architekturentwicklungsprozess als iterativ-zyklische Methode zur aktiven Gestaltung von Architekturen. Wie in Abbildung 13 dargestellt, liefert die Architecture Development Method die Planung der Architektur basierend auf den individuellen Anforderungen für den jeweiligen Fokusbereich.

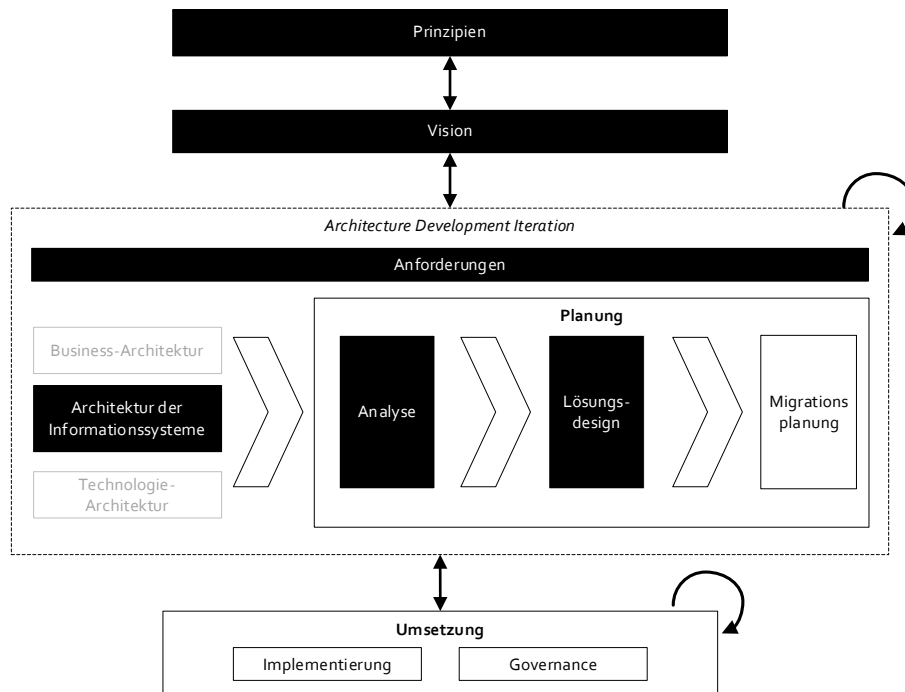


Abbildung 13: Einbettung des GAP-Modells in die Iterationszyklen der TOGAF Architecture Development Method
(Quelle: Eigene Darstellung in Anlehnung an The Open Group [9, Seite 1–5])

Ausgangspunkt für die TOGAF Architecture Development Method ist die Definition grundsätzlicher Prinzipien. Dieser in TOGAF als „Preliminary“ bezeichnete Schritt umfasst neben der Etablierung gemeinsamer Standards und Methoden insbesondere die Analyse betrieblicher, technischer und organisatorischer Grundsätze und Gegebenheiten. [51, Seite 16] Als Nächster Schritt bei der Architekturentwicklung beschreibt TOGAF die Definition der Vision. Hierbei werden die langfristigen Ziele beschrieben, welche mit der jeweiligen Architektur erreicht werden sollen. [51, Seite 30] In Kombination mit individuellen Anforderungen können nun die Anforderungen an die Architektur definiert werden. Diese Anforderungen bilden den zentralen Bezugspunkt bei der weiteren Entwicklung der Architektur. [51, Seite 130] Die Architektur der Informationssysteme wird dabei iterativ entworfen, weiterentwickelt und umgesetzt. Hierbei ist hervorzuheben, dass die Architekturentwicklung in Wechselwirkung mit Implementierung und Governance steht, jedoch zeitlich entkoppelt betrachtet werden kann, so dass im Sinne einer vorausschauenden Planung mehrere aufeinander aufbauende Ausbaustufen geplant werden können, ohne dass hierbei eine sofortige Implementierung des jeweiligen Lösungsdesigns erfolgen muss. [9, Seite 4]

2.4.4. Standardisierte Architekturentscheidungen mit der TOGAF Architecture Alternatives Method

TOGAF definiert den Architekturauswahlprozess als „Architecture Alternatives Method“ in dem zwischen verschiedenen alternativen Architekturausprägungen abgewägt wird. [9, Seite 74] Dieser Prozess ist in Abbildung 14 dargestellt.

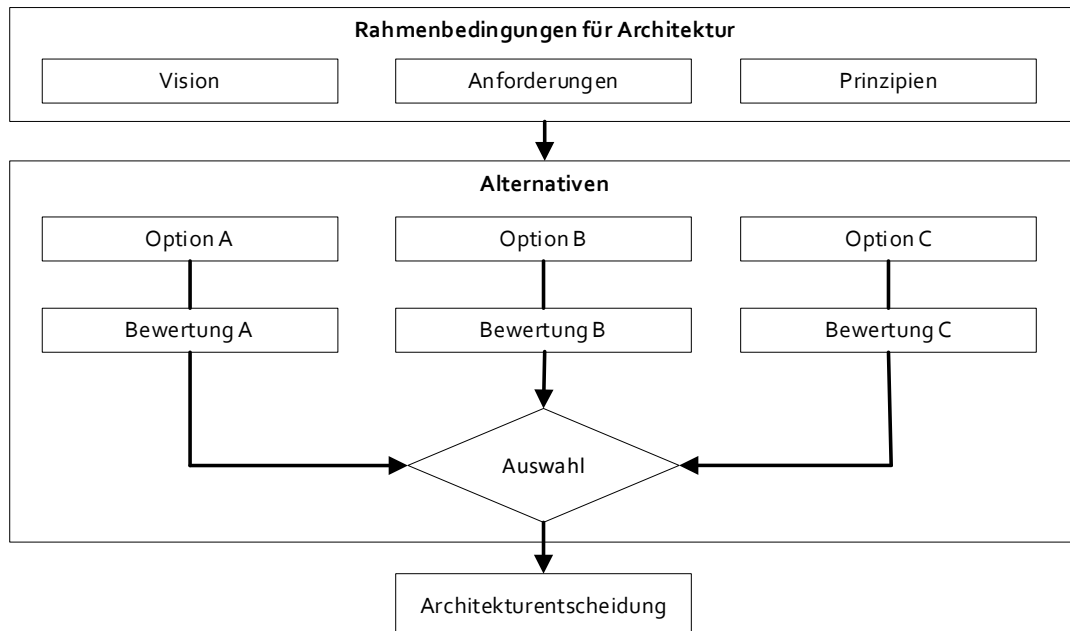


Abbildung 14: Architekturentscheidungsprozess nach TOGAF
(Quelle: Eigene Darstellung in Anlehnung an The Open Group [9, Seite 74])

Bestimmend für die Architekturentscheidung sind die Rahmenbedingungen für die Architektur, definiert durch Vision, Anforderungen und Prinzipien. In der TOGAF Architecture Alternatives Method wird zunächst ein Lösungsraum von potenziellen Architekturentwürfen generiert, welche dann jeweils anhand der architektonischen Rahmenbedingungen bewertet werden. Anschließend erfolgt die Auswahl der besten Lösung durch Abwägung von Vor- und Nachteilen der jeweiligen Alternativen. [9, Seite 73–76]

2.5. Zusammenfassung

„APIs sind der „Kleber“, der die Elemente einer Systemarchitektur zusammenhält.“

So beschrieben Perry und Wolf in Ihrem Beitrag „Foundations for the Study of Software Architecture“ 1992 die Rolle der Elemente einer Systemlandschaft, welche für die Integration der verschiedenen Systeme genutzt werden. [33] Der Begriff des Konnektivitätsökosystems beschreibt dabei die Gesamtheit aller verbundenen Systeme, welche über APIs Daten und Funktionalität miteinander austauschen. [3] Akteure in diesem Ökosystem lassen sich als Provider, Consumer und Prosumer kategorisieren. [31, Seite 10–11]

Die IT-Architektur definiert die fundamentalen Konzepte, Prinzipien und Eigenschaften nach denen diese Systemlandschaft gestaltet ist [47, Seite 4] Da eine leistungsfähige digitale Infrastruktur Grundlage für die digitale Transformation ist, kommt der zweckdienlichen Gestaltung von Integrationsarchitekturen eine besondere Rolle zu. [23, Seite 51]

Das Bundesministerium für Wirtschaft und Klimaschutz definiert Digitalisierung als „Verwendung von Daten und algorithmischen Systemen für neue und verbesserte Prozesse, Produkte und Geschäftsmodelle“ [2] Der Branchenverband Bitkom beschreibt Digitalisierung deckungsgleich als den Einsatz digitaler Technologien zur Steigerung von Effizienz und Produktivität. [14] Durch die Digitalisierung von Geschäftsprozessen eröffnen sich insbesondere mit der Vernetzung, Virtualisierung und intelligenten Datenanalyse völlig neue Wertschöpfungsmöglichkeiten. [23, Seite 50]

Zwar werden bereits Digitalisierungspotentiale genutzt, jedoch bremsen fehlende Expertise und begrenzte Ressourcen die digitale Transformation. [4, Seite 38]. Aus diesem Grund erodiert der Autor modellbasierte Ansätze zur Optimierung von Integrationsarchitekturen. Der Autor stützt sich dabei auf das etablierte „TOGAF“ Architekturrahmenwerk. Architekturen können nach TOGAF auf verschiedenen Granularitätsebenen betrachtet werden, wobei mit tieferer Detailtiefe eine Segmentierung in Teilbereiche stattfindet. [9, Seite 25]

Die **Enterprise-Ebene** enthält die Gesamtheit aller IT-Systeme und -Anwendungen innerhalb einer Organisation und stellt damit den größten Detailgrad der Betrachtung dar. Eine Organisation, bzw. ein „Enterprise“ ist dabei als ein abgrenzbarer Zusammenschluss verschiedener Organisationseinheiten mit einem gemeinsamen Zweck definiert. [6, Seite 2]

Die **Applikationsebene** beschreibt die Architektur mit Fokus auf einzelne Applikationen innerhalb der Systemlandschaft und definiert somit ein ausgewähltes Segment der Enterprise-Architektur in einem höheren Detailgrad. Die Komplexität der einzelnen Applikationen und ihrer APIs lässt sich damit kapseln und für den jeweiligen Detailgrad abstrahieren. [9, Seite 21]

Die **Technische Ebene** beschreibt die Lösungsarchitektur, bzw. „Solution-Architecture“ und definiert die technische Implementierung einer bestimmten Komponente innerhalb der Systemlandschaft. [9, Seite 20–25]

Diese Architekturen lassen sich dann mithilfe der Unified Modeling Language (UML) präzise, standardisiert und maschinenlesbar abbilden und modellieren. [49, Seite 43]

Die Rahmenbedingungen für die Architektur ergeben sich aus der Architekturvision, den spezifischen Anforderungen, sowie den zugrundeliegenden Prinzipien. Die Gestaltung der IT-Architektur erfolgt in der Architektur-Realisierung durch Planung und Umsetzung von Architekturentscheidungen. Basierend auf einer Zustandsanalyse und Bewertung jeweiligen Chancen und Risiken entsteht im Kontext der IT-Architektur ein individuelles Lösungsdesign, sowie eine Migrationsplanung, welche den Plan für die Transformation beschreibt. Diese Architekturentscheidungen werden dann über anwendungsspezifische Implementierung und allgemeine Governance umgesetzt. [48, Seite 5–6]

3. Stand der Forschung und Analyse existierender Ansätze zur Klassifizierung von Integrationsarchitekturen

In diesem Kapitel wird ausgehend vom aktuellen Stand der Forschung eine Taxonomie zur Klassifizierung von Integrationsarchitekturen erarbeitet, womit die Forschungsfrage FF 2 (*Nach welcher Systematik können Integrationsarchitekturen unterschieden werden und welche Klassifizierung ergibt sich daraus?*) beantwortet wird.

Der Autor nutzt die in Abschnitt 2.3 beschriebenen Ebenen der IT-Architektur und Systematik, nach der Designartefakte innerhalb der entsprechenden Granularitätsebenen anhand ihrer Merkmale segmentiert werden können:

- **Enterprise-Ebene:** Klassifizierung von Integrationslandschaften nach Topologie
- **Applikationsebene:** Klassifizierung von Integrationsmuster nach Paradigma
- **Technische Ebene:** Klassifizierung von Integrationsverfahren nach Methode

3.1. Beschreibung des analytischen Vorgehens

Eine trennscharfe Klassifizierung von Integrationsarchitekturen ist Grundlage für die Vergleichbarkeit von Designentscheidungen und bildet die Basis für das Bewertungsmodell von Integrationsarchitekturen. Hierfür definiert der Autor eine Taxonomie, anhand derer sich jede beliebige Integrationsarchitektur klassifizieren lässt. Hierfür werden Artefakte der Architektur auf den drei in Abschnitt 2.3 beschriebenen Granularitätsebenen analysiert und segmentiert.

Die Segmentierung der Architekturen auf den verschiedenen Granularitätsebenen erfolgt nach dem aktuellen Stand der Forschung im Bereich der IT-Architektur, welche vom Autor im Folgenden erfasst und kontextualisiert wird. Entscheidend für die Klassifizierung sind dabei identifizierter Merkmale, welche die jeweiligen Ausprägungen verschiedener Integrationsarchitekturen auf der jeweiligen Ebenen teilen bzw. unterscheiden. Anhand dieser Merkmale erfolgt dann eine Gruppierung von Integrationsarchitekturkomponenten in Klassen und eine Beschreibung derselben.

Aus der Kombination der Klassifizierung auf den drei Granularitätsebenen ergibt sich ein Architekturarchetyp, welcher der betrachteten Integrationsarchitektur zugeordnet werden kann. Mithilfe dieser Archetypisierung von Integrationsarchitekturen können diese dann standardisiert eingeordnet und vergleichbar bewertet werden.

3.2. Enterprise-Ebene: Klassifizierung von Integrationslandschaften nach Integrationstopologie

Segmentierung von Integrationsarchitekturen auf Enterprise-Ebene

Die Integrationslandschaft beschreibt, welche Akteure wie im Konnektivitätsökosystem miteinander verbunden sind. Die konzeptionelle Schematik, nach welcher diese Systemintegrationen realisiert sind und Informationen zwischen den verknüpften Systemen ausgetauscht werden, wird als Integrationstopologie beschrieben. Die Auswahl der Topologie hat dabei Einfluss auf die Kontrolle über den Informationsfluss, sowie die Ausfallsicherheit und Skalierbarkeit des Ökosystems. [8, Seite 24–28]

Wie in Abschnitt 2.4.1 beschrieben, werden Systeme als Knoten und die Integrationen als Kanten modelliert. Integrationslandschaften lassen sich somit als Netzwerke darstellen und anhand ihrer Topologie kategorisieren. Hierfür wendet der Autor etablierte Merkmale aus der technologieagnostischen Klassifizierung von Verbindungsnetzwerken an.

Das Merkmal **Kontrollstrategie** beschreibt, ob eine Steuerung über eine zentrale Komponente im Konnektivitätsökosystem erfolgt. Ist dies der Fall, so ist das Merkmal als *zentral* ausgeprägt. Fehlt diese Komponente, ist die Topologie *dezentral*. [7, Seite 41]

Das Merkmal **Kommunikationsflexibilität** beschreibt, wie viele Kommunikationspartnerschaften ein System eingeht. Wird eine Information ausschließlich über einen Pfad ausgetauscht, so ist das Merkmal als *exklusiv* ausgeprägt. Werden Informationen über mehrere Pfade im Ökosystem weitergereicht, so ist das Merkmal als *distributiv* ausgeprägt. [7, Seite 30]

Abbildung 15 zeigt die Kategorisierung nach den oben beschriebenen Merkmalen. dargestellt in vier Kategorien **Hub**, **Bus**, **Tree** und **Mesh** einordnen.

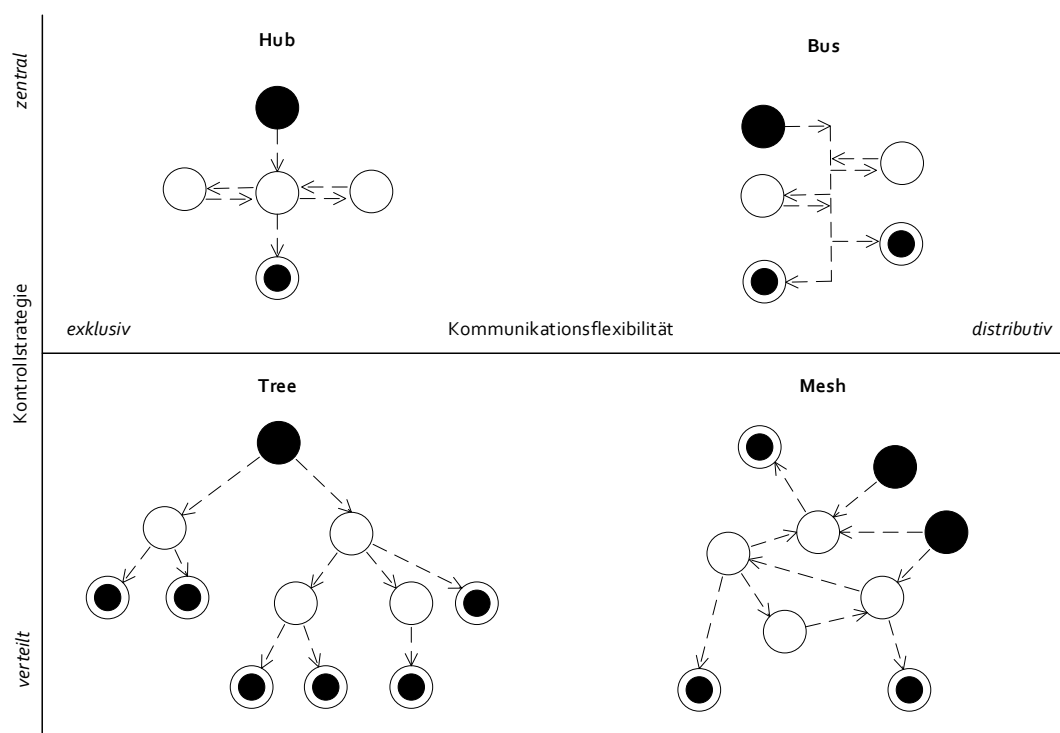


Abbildung 15: Kategorisierung von Integrationstopologien

Die verschiedenen Topologien schließen sich nicht aus, sondern können innerhalb eines Ökosystems je nach Anwendungsfall kombiniert werden. Ringstrukturen sind explizit ausgenommen, da Informationsflüsse mit Start und Ende modelliert werden und es somit Ringtopologien in einer Endlosschleife resultieren würden.

3.2.1. Integration-Hub

Merkmale: Zentrale Kontrollstrategie, exklusive Kommunikationsflexibilität

Eine Hub-Topologie (alternativ: Stern-Topologie) zeichnet sich durch die zentrale Kontrollstrategie, sowie exklusive Kommunikationsflexibilität aus. Alle Systeme sind mit dem Hub als zentrale Komponente verbunden und tauschen ausschließlich über diese Informationen aus (siehe Abbildung 16). Provider, Consumer, sowie Prosumer nutzen dabei typischerweise dasselbe Datenmodell. [8, Seite 26]

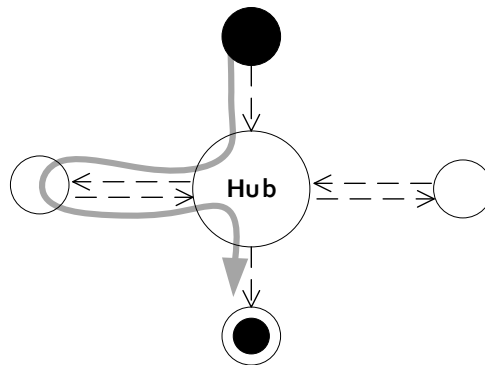


Abbildung 16: Informationspfad in einer Hub-Topologie

3.2.2. Integration-Bus

Merkmale: Zentrale Kontrollstrategie, distributive Kommunikationsflexibilität

Eine Bus-Topologie (alternativ: „Datenautobahn“) zeichnet sich durch die zentrale Kontrollstrategie, sowie distributive Kommunikationsflexibilität aus. Alle Systeme kommunizieren über eine zentrale Middleware miteinander und tauschen über diese Informationen direkt miteinander aus (siehe Abbildung 17). Im Vergleich zum Integration-Hub ist diese zentrale Komponente jedoch kein aktiver Akteur der Integration. [8, Seite 24]

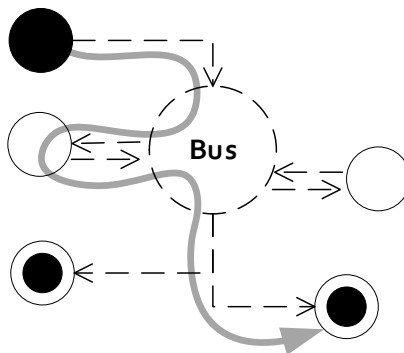


Abbildung 17: Informationspfad in einer Bus-Topologie

3.2.3. Integration-Tree

Merkmale: Verteilte Kontrollstrategie, exklusive Kommunikationsflexibilität

Eine Tree-Topologie (alternativ: Baum-Topologie) zeichnet sich durch eine verteilte Kontrollstrategie, sowie exklusive Kommunikationsflexibilität aus. Systeme sind ohne zentrale Komponente direkt miteinander verbunden. Informationen werden dabei über genau einen Pfad direkt mit den Systemen geteilt. Die Kommunikation erfolgt für ein Datenobjekt ausschließlich in eine Richtung. Im Gegensatz zum Integration-Mesh gibt es keine alternativen, bzw. redundanten Kommunikationspfade. Eine Zusammenführung derselben Informationen aus verschiedenen Systemen erfolgt demnach nicht. (siehe Abbildung 18). [8, Seite 27]

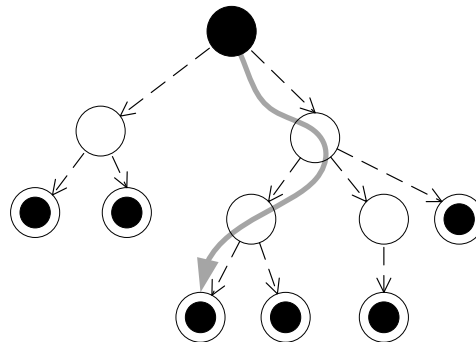


Abbildung 18: Informationspfad in einer Tree-Topologie

3.2.4. Integration-Mesh

Merkmale: Zentrale Kontrollstrategie, distributive Kommunikationsflexibilität

Eine Mesh-Topologie (alternativ: Netzwerk-Topologie) zeichnet sich durch eine verteilte Kontrollstrategie, sowie distributive Kommunikationsflexibilität aus. Systeme sind direkt miteinander verbunden. Informationen werden auf verschiedenen Pfaden übermittelt und verarbeitet. Informationspfade können im Integration-Mesh wieder zusammengeführt werden (siehe Abbildung 19). Charakteristisch ist das Fehlen einer zentralen Komponente, welche für die Verwaltung der Kommunikation zuständig wäre. Stattdessen treten alle Akteure mit beliebig vielen anderen Akteuren in Kontakt und tauschen Informationen über verschiedene alternative Routen aus. [8, Seite 26–27]

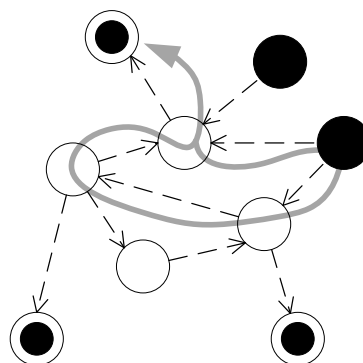


Abbildung 19: Informationspfad in einer Mesh-Topologie

3.3. Applikationsebene: Klassifizierung von Integrationsmustern nach Integrationsparadigma

Segmentierung von Integrationsarchitekturen auf Applikationsebene

Ein Integrationsmuster beschreibt auf konzeptioneller Ebene, wie Systeme miteinander verbunden werden. Jedem Integrationsmuster liegt ein bestimmtes Paradigma zugrunde. Das Integrationsparadigma beschreibt auf konzeptioneller Ebene die Gestaltung der Kommunikation zwischen integrierten Systemen.

Es gibt verschiedene Attribute zur technologieagnostischen Kategorisierung und strukturierten Einordnung von Architekturmustern. So kann zunächst unterschieden werden, wie Integrationsaktivitäten initiiert werden und wie hoch der Grad der Virtualisierung ist. Hinzu kommen die Strukturierung und *Verteilung des Datenmodells*, auch der *Grad der Kopplung* als differenzierende Merkmale. [10]

Das Merkmal **Initiierung des Informationsaustausches** beschreibt die Art und Weise, wie der Informationsaustausch zwischen Consumer und Provider ausgelöst wird. Hierbei kann zwischen einer proaktiven (pull) und einer reaktiven (push) Initiierung unterschieden werden. Bei einer proaktiven Kommunikation erfolgt die Transaktion als sofortige Reaktion auf ein Ereignis (push), reaktive Kommunikation hingegen fordert den aktuellen Informationsstand zu gegebenem Zeitpunkt an (pull). [12, Seite 45]

Das Merkmal **Ausführung der Informationsintegration** beschreibt den Grad der Virtualisierung und, bzw. ob die übertragenen Daten persistiert werden. Bei einer materialisierenden Integration werden Daten physisch kopiert und es entsteht ein Duplikat des ursprünglichen Datums. Bei virtualisierten Integrationen verbleiben die Daten im Quellsystem und werden lediglich zur unverzüglichen Verarbeitung an das Zielsystem übertragen. [10]

Das Merkmal der **Kopplung** beschreibt, wie fest die integrierten Systeme miteinander verbunden sind. Bei einer losen Kopplung können Systeme leicht aus dem Verbund entfernt und andere hinzugefügt werden. Dies ist bei einer festen Kopplung nur eingeschränkt möglich. [10]

Das Merkmal der **Strukturierung** und Verteilung des Datenmodells beschreibt, inwieweit ein gemeinsames Datenmodell für die Integration genutzt wird. [10]

Da sowohl der Grad der Kopplung, sowie die Nutzung eines verteilten, oder zentralen Datenmodells von der jeweiligen Implementierung der Integration abhängen, erfolgt die Kategorisierung ausschließlich auf Grundlage von den Merkmalen Initiierung und Virtualisierung. Beide Merkmale gemeinsam beschreiben im Grunde, wie die Transaktionen im Konnektivitätsökosystem abgewickelt werden.

Im Rahmen dieser Arbeit werden Architekturmuster auf Grundlage von Gemeinsamkeiten in Bezug auf die beschriebenen Merkmale *Initialisierung des Informationsaustausches* und *Ausführung der Informationsintegration* klassifizieren. Integrationsmuster lassen sich dabei folgenden Paradigmen zuordnen:

- **Storage-Driven:** materialisierten Integration mit proaktiver Initialisierung
- **Service-Driven:** virtuellen Integration mit einer proaktiven Initialisierung
- **Event-Driven:** Integration mit einer reaktiven Initialisierung

Abbildung 20 zeigt die Klassifizierungslogik für die Zuordnung von Integrationsmustern zu Paradigmen. Auf eine weitere Segmentierung reaktiver Integrationen wird dabei verzichtet, da eine Unterscheidung zwischen materialisierter und virtualisierter Integration bei eventgetriebenen Architekturen keine relevanten Unterschiede bei der Bewertung der Integrationsarchitektur ergeben würde.

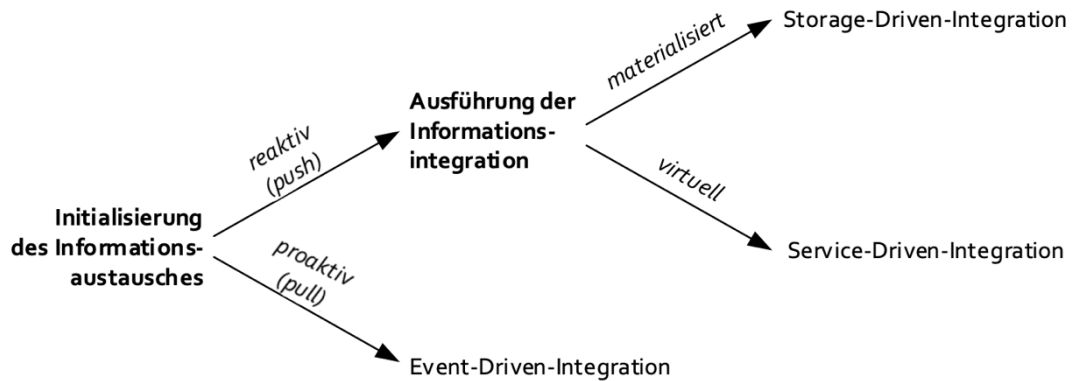


Abbildung 20: Klassifizierung von Integrationsarchitekturmustern

Abbildung 21 zeigt eine beispielhafte Klassifizierung von Integrationsansätzen nach der beschriebenen Systematik. So können Event-Driven-Integrations beispielsweise Implementierungen mit Data Streams, Event-Hubs, Event-Messaging, aber auch Remote Procedure Calls (RPC) klassifiziert werden. Implementierungen mit Enterprise Data Bus (EDB), Batch File Transfers, ETL-Jobs und Data Warehouses sind klassischerweise als Storage-Driven-Integrations ausgeprägt. Service-Driven-Integrations umfassen unter anderem Implementierungen mit Microservices, Service Orientated Architecture (SOA), Enterprise Service Bus (ESB), sowie Webservices. Die Auswahl der Integrationsansätze basiert auf den Antworten von Chat-GPT auf die Frage nach bekannten Architekturmustern für Systemintegration [Anhang 7].

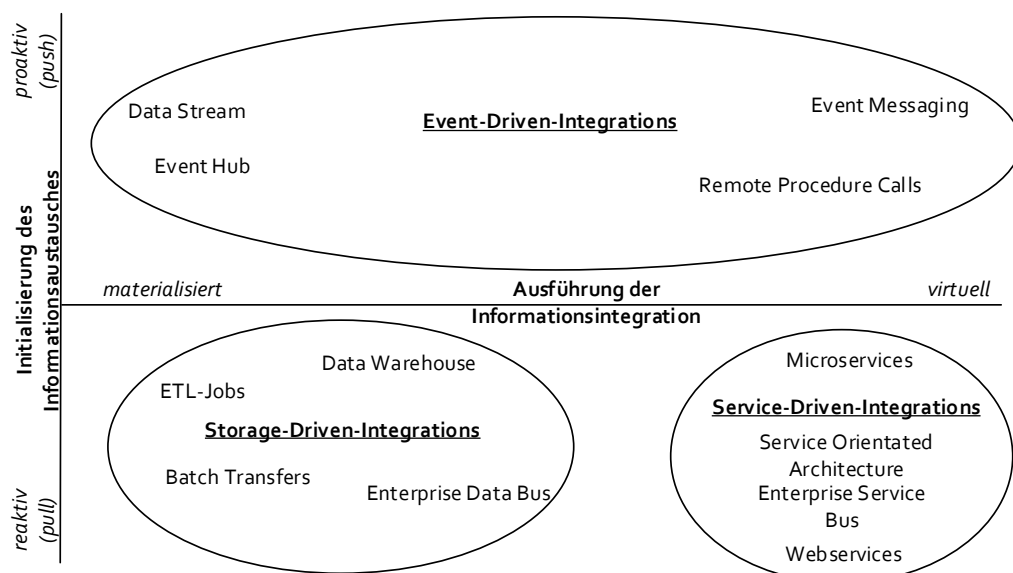


Abbildung 21: Beispielhafte Klassifizierung von Integrationsparadigmen

Eine zentrale Erkenntnis der erfolgten Klassifizierung ist, dass die genutzte Technologie nicht zwangsläufig deckungsgleich mit dem entsprechenden Integrationsparadigma ist. So können durchaus Webservices genutzt werden, um Daten im Rahmen einer Storage-Driven-Integration aus einem System zu extrahieren und in ein anderes System zu übertragen. Auch können Event-Driven-Integrations mit Microservices realisiert werden, wenn beispielsweise eine Aktivität in einem System einen Microservice auslöst.

3.3.1. Storage-Driven-Integration

Merkmale: Proaktive Initialisierung des Informationsaustausches, materialisierte Ausführung der Informationsintegration

Unter der Klasse *Storage-Driven-Integration* vereinen sich alle Integrationen, bei denen Daten zu einem bestimmten Zeitpunkt physisch kopiert werden (*materialisierte Ausführung der Informationsintegration*). Die Kommunikation bei speichergetriebenen Integrationen erfolgt auf proaktive Anfrage durch den Consumer, oder die Middleware (*proaktive Initialisierung des Informationsaustausches*). [52]

Benötigte Komponenten sind Provider, Consumer und eine Middleware, welchen typischerweise einen ETL-Prozess ausführt. ETL steht als Akronym für Extract, Transform, Load. Dabei werden in bestimmten Zeitabständen die Daten aus den Quellsystemen in die Zielsysteme übertragen [53, Seite 45–46]

Provider sind als Quellsysteme integriert und erlauben der Middleware über eine Schnittstelle lesenden Zugriff auf die benötigten Daten. Alternativ kann der Provider die zu integrierenden Daten auch aktiv an die Middleware übertragen.

Consumer sind als Zielsystem integriert und gewähren der Middleware schreibenden Zugriff auf das System. Alternativ können Consumer auch auf die Middleware zugreifen und die Daten in ihr System kopieren.

Die **Middleware** ist das zentrale Element welche als aktive Komponente alle Aktivitäten des Integrationsprozesses ausführt.

Wie in Abbildung 22 dargestellt, greift die Middleware zunächst auf den Provider, bzw. das Quellsystem zu und extrahiert die zu integrierenden Daten (Extract). Hierbei werden typischerweise die Daten aus dem Quellsystem ohne Formatänderung zur weiteren Verwendung in der Middleware abgelegt. [31, Seite 22]

Im Nächsten Schritt (Transform) wandelt die Middleware die Quelldaten in das Zielformat um und bereitet diese für die weitere Verarbeitung vor. Diese Transformationen können je nach Integrationstopologie entweder zentralisiert erfolgen, indem beispielsweise ein globales Datenmodell als Referenzstandard genutzt wird, oder es wird im Sinne der verteilten Topologie direkt auf die Zielsysteme gemappt. [31, Seite 23]

Im letzten Schritt des ETL-Prozesses (Load) werden die Daten aktiv in das Zielsystem geschrieben, bzw. an den Consumer übermittelt und stehen dort zur weiteren Verarbeitung zur Verfügung. Die Synchronisierung der Daten kann im einfachsten Fall auf Grundlage eines kompletten Exportes realisiert werden. Effizientere Implementierungen nutzen Zeitstempel, oder Delta-Logs um nur die Änderungen seit dem letzten ETL-Zyklus zu synchronisieren. [53, Seite 43–49]

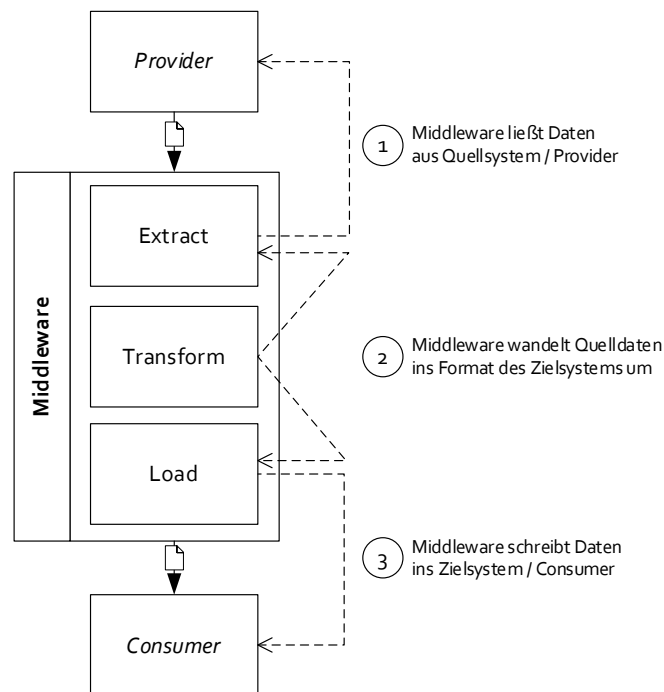


Abbildung 22: schematische Darstellung eines ETL-Prozesses

Storage-Driven-Integration kann sowohl zentralisiert als auch verteilt implementiert werden. In einer verteilten Topologie realisiert die Middleware die direkte Kommunikation zwischen zwei Systemen.

Beispiel für eine zentralisierte Storage-Driven-Integration ist ein Enterprise Data Bus (EDB), welcher über verschiedene Schnittstellen (bspw. FTP, ODBC, HTTP/APIs) Zugriff auf die verbundenen Systeme hat.

In Abbildung 23 ist dargestellt wie mithilfe einer ETL-Implementierung Daten von Provider-Systemen in Consumer-Systeme kopiert werden. Der Datenaustausch kann dabei sowohl nach Push- als auch nach Pull-Pattern implementiert sein. Data-Warehouse-Systeme sind ein weiterer klassischer Anwendungsfall für Storage-Driven-Integration. Hier werden Daten über einen größeren Zeitraum gesammelt und analysiert.

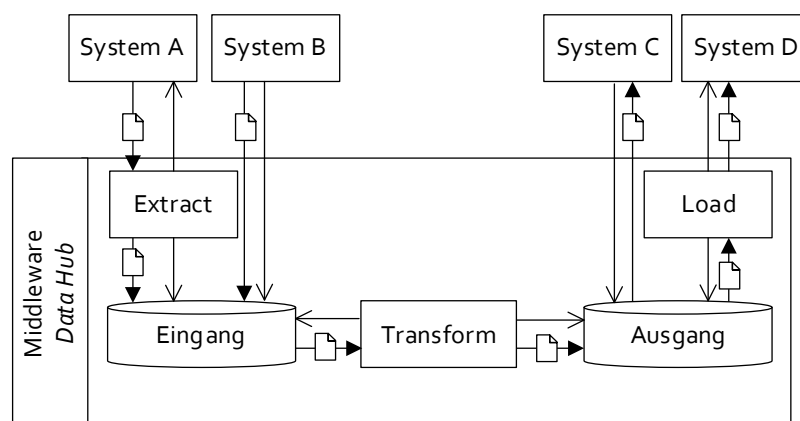


Abbildung 23: Data Hub als Beispiel für Storage-Driven-Integration

3.3.2. Service-Driven-Integration

Merkmale: Proaktive Initialisierung des Informationsaustausches, virtuelle Ausführung der Informationsintegration

Unter der Klasse *Service-Driven-Integration* vereinen sich alle Integrationen, bei denen Daten und Funktionalitäten als Services gekapselt werden. Dabei werden Informationen immer referenziert und nicht im Consumer-System persistiert (*virtuelle Ausführung der Informationsintegration*). Die Kommunikation bei servicegetriebenen Integrationen erfolgt auf Anfrage, welche vom Provider beantwortet wird (*proaktive Initialisierung des Informationsaustausches*), also nach Request-Reply-Muster. [54, Seite 121]

Systeme stellen Daten und Funktionalitäten als Service innerhalb des Konnektivitätsökosystems zur Verfügung. Consumer haben Zugriff auf Provider, ohne dass Daten zwingend im konsumierenden System persistiert werden müssen. Dieser Zugriff kann sowohl direkt aus den Systemen als auch mithilfe einer Middleware realisiert werden. [55]

Provider bieten Daten und Funktionalität des Systems als Service an

Consumer nutzen angebotene Services

Middleware wird zur Optimierung der Integrationen genutzt.

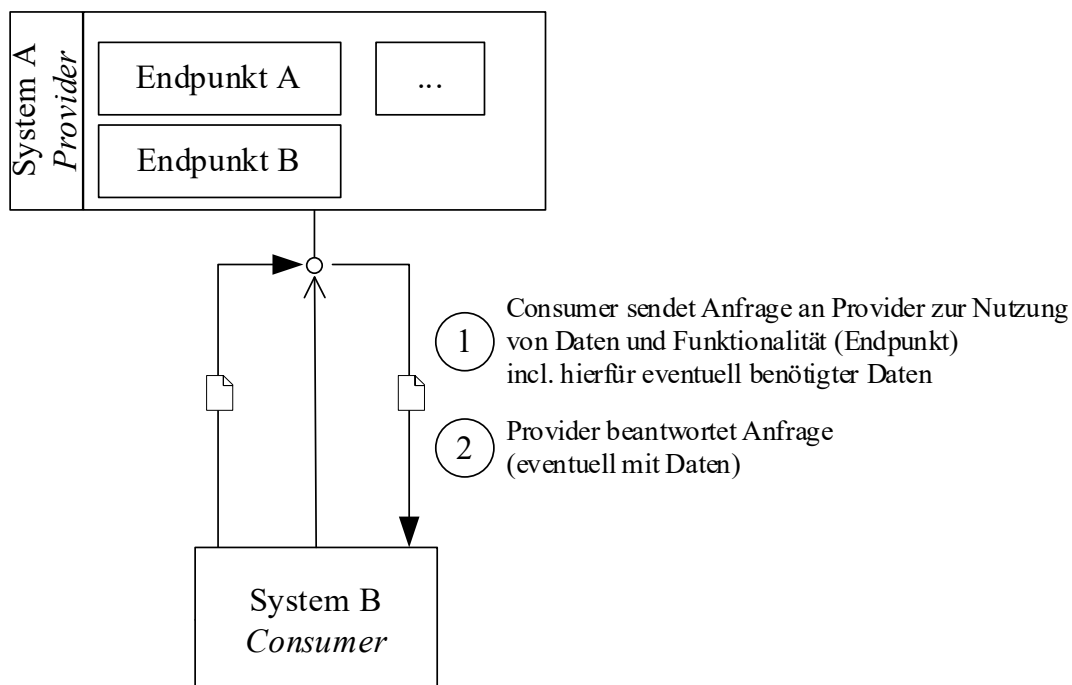


Abbildung 24: Schematische Darstellung einer Servicebasierten Integrationsprozesses

Wie in Abbildung 24 dargestellt, stellt der Provider eine Auswahl seiner Services als Endpunkte über eine Schnittstelle bereit. Der Consumer sendet eine Anfrage an den Provider (Request). Dabei werden alle für die Ausführung des Service benötigten Daten vom Consumer zum Provider übertragen (Payload). Nachdem der Provider die Anfrage intern bearbeitet hat, reagiert er auf die Anfrage und sendet eine Antwort zurück (Response). Diese Rückantwort enthält die Rückmeldung des Service, beispielsweise ein angefragtes Datenobjekt, oder die Bestätigung über die Ausführung einer Aktion. [56]

Services verbergen für die Consumer die Komplexität der Implementierung, da anstatt der gesamten Datenstruktur lediglich die für den Service relevanten Strukturen exponiert werden. Somit wird eine Entkopplung von Integration und dahinterliegender Systemfunktionalität erreicht. Interne Änderungen des Providers sind ohne Auswirkungen auf die verbundenen Consumer. Auch können APIs zunächst als Mock zur Verfügung gestellt werden, ohne dass die dahinter liegende Systemfunktionalität bereits ausimplementiert wurde. Die Implementierung einer API kann dabei beliebig häufig angepasst werden, solange die Schnittstelle selbst unverändert, bzw. abwärtskompatibel bleibt.

Services können einerseits als Direktverbindungen im Rahmen einer verteilten Integrationstopologie frei mit allen relevanten Akteuren im Konnektivitätsökosystem verbunden werden. Andererseits kann auch entsprechend der zentralisierten Topologie für die Integration beispielsweise ein zentraler Service-Hub oder ein zentrales API-Gateway genutzt werden. Bei komplexeren Strukturen bietet sich der Einsatz einer Middleware zum API-Management an.

3.3.3. Event-Driven-Integration

Merkmal: Proaktive Initialisierung des Informationsaustausches

Unter der Klasse *Event-Driven-Integration* vereinen sich alle Integrationen, die unmittelbar durch eine Änderung der Informationen im Provider-System initiiert werden (*proaktive Initialisierung des Informationsaustausches*). Dabei ist es unerheblich, ob die verteilten Informationen im weiteren Verlauf persistiert werden (*Ausführung der Informationsintegration*).

Die Systemintegration erfolgt nach dem Publish-Subscribe-Architekturmuster. [57, Seite 6] Systeme senden Nachrichten aus, um beteiligte Akteure im Konnektivitätsökosystem über Aktivitäten zu informieren. Diese Nachrichten (Message) können Themen (Topics) zugeordnet werden, wobei mehrere Systeme Messages aus demselben Topic beitragen können. Zudem können beliebig viele Consumer-Systeme auf Messages in einem Topic reagieren (Subscriber). [58]

Provider senden als Publisher, bzw. „Event Producer“ Nachrichten aus. Je nach Event werden Messages und die Topics gesendet. Provider-Systeme informieren somit das Konnektivitätsökosystem proaktiv über Änderungen. [59, Seite 7]

Consumer empfangen als Subscriber, bzw. „Event Consumer“ alle Nachrichten der Topics, welche sie abonniert haben. Dabei sendet die Middleware aktiv eine Nachricht an die verbundenen Systeme aus. [59, Seite 7]

Middleware agiert als Vermittler zwischen Consumer und Provider und stellt die Infrastruktur für die Nachrichtenabwicklung. Diese kann in diesem Kontext auch als „Event Broker“ bezeichnet werden. [59, Seite 5–7]

Abbildung 25 zeigt einen schematischen Integrationsprozess nach eventgetriebenem Paradigma. Der Provider informiert die Middleware aktiv über ein Event, indem eine Message an ein Topic gesendet wird. Ein Provider kann dabei Messages an beliebig viele Topics senden, welche wiederum Nachrichten von beliebig vielen Providern erhalten können. Die Middleware verwaltet diese Nachricht und informiert alle Consumer, welche das jeweilige Topic abonniert haben. Ein Topic kann von beliebig vielen Consumern abonniert werden. Ein Consumer kann beliebig viele Topics abonnieren. Die Consumer, bzw. Subscriber erhalten mit der Message alle benötigten Informationen über die erfolgte Änderung und können nun die entsprechenden Aktivitäten einleiten und auf das Event reagieren. [59, Seite 13–14]

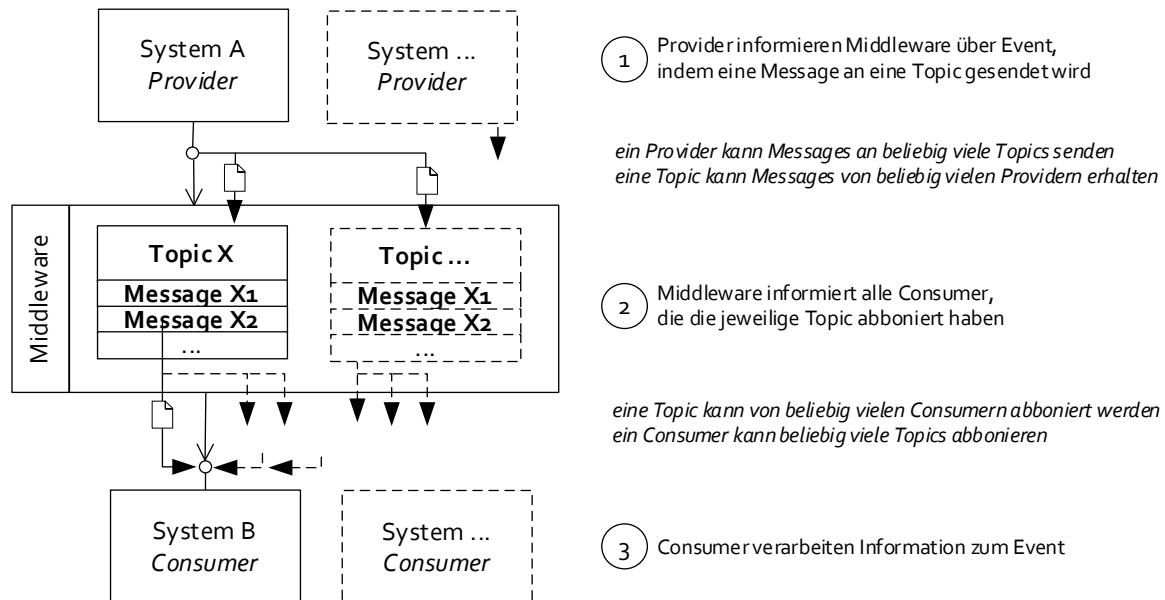


Abbildung 25: Schematische Darstellung eines eventbasierten Integrationsprozesses

Das oben beschriebene Paradigma setzt voraus, dass alle Consumer zu jeder Zeit auf ein Event reagieren können. Hierdurch würden weniger Performante Akteure die Performance des gesamten Konnektivitätsökosystems beschränken. Um eine Entkopplung der Provider und Consumer zu erreichen, werden Messages nicht direkt weiterkommuniziert. Stattdessen werden Nachrichten wie in Abbildung 26 schematisch dargestellt in einer Warteschlange (Queue) zwischengespeichert, aus welcher sie dann zu gegebener Zeit für die Bearbeitung durch den Consumer entnommen werden können. [59, Seite 12]

In einer Event-Driven-Integration mit Message-Queue informiert der Provider wie oben beschrieben die Middleware mit einer Nachricht über eine Änderung. Nachdem die Message in dem entsprechenden Topic registriert wurde, verteilt die Middleware die eingegangene Nachricht an eine oder mehrere Queues. [59, Seite 14] So kann zum Beispiel jedes angebundene System eine eigene Queue erhalten, in die für dieses System relevante Messages eingeordnet werden. Die Message wird so lange in der Queue zwischengespeichert, bis der Consumer bzw. Subscriber auf die Benachrichtigung reagiert und die Nachricht aus der Queue entnimmt [60]. Die Queue kann dabei verschiedene Eigenschaften haben. So kann beispielsweise entweder ein Push- oder Pull-Pattern, oder eine bestimmte Sortierung der Nachrichten implementiert werden. [59, Seite 8]

Eine Event-Driven-Integration setzt aufgrund der benötigten Middleware ein zentrales Element und folgt somit tendenziell der zentralisierten Topologie mit Event-Hub. Jedoch kann auch ein verteilter Ansatz mit vielen unterschiedlichen eventbasierten Integrationen als Prosumer implementiert werden. Die übertragenen Daten werden dabei in jedem Fall kurzzeitig persistiert, können aber nach Abarbeitung des Events verworfen werden.

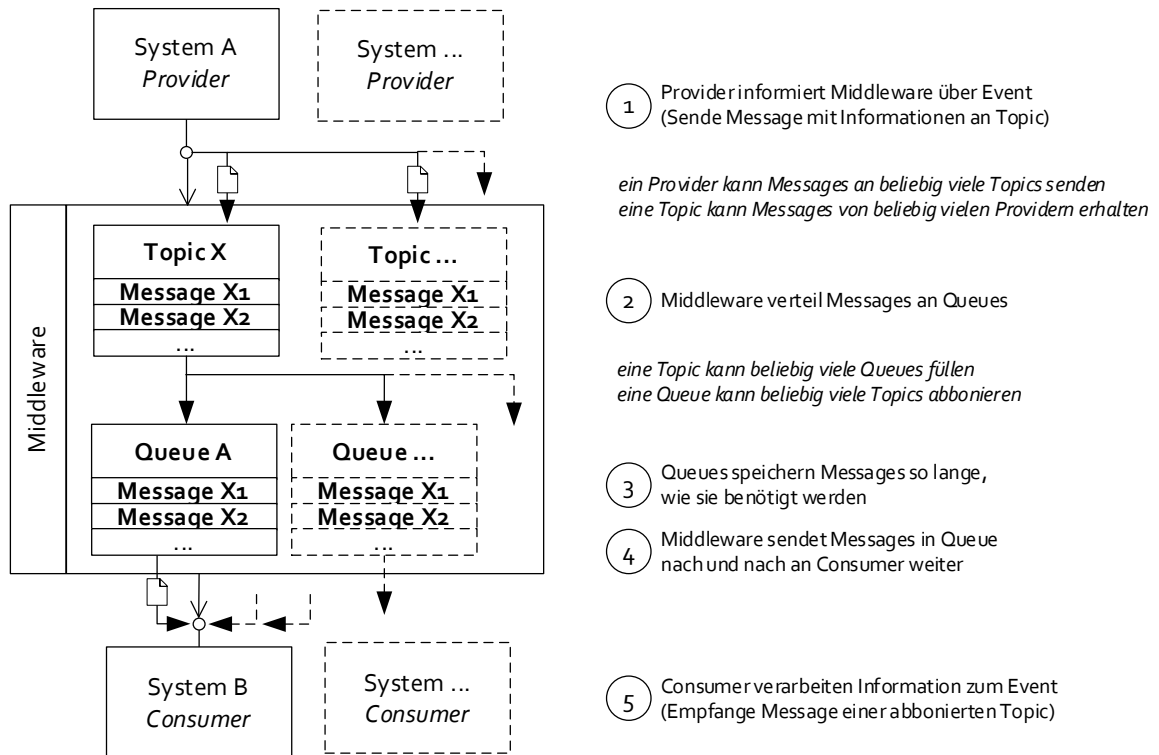


Abbildung 26: Schematische Darstellung eines eventbasierten Integrationsprozesses mit Event-Queue

Event-Driven-Integration findet beispielsweise beim Streaming von Sensordaten im IoT-Bereich Anwendung (siehe Abbildung 27). So können zum Beispiel Sensordaten von Fahrzeugen als MQTT-Messages in Echtzeit übertragen und von verschiedenen Systemen weiterverarbeitet werden. [58]

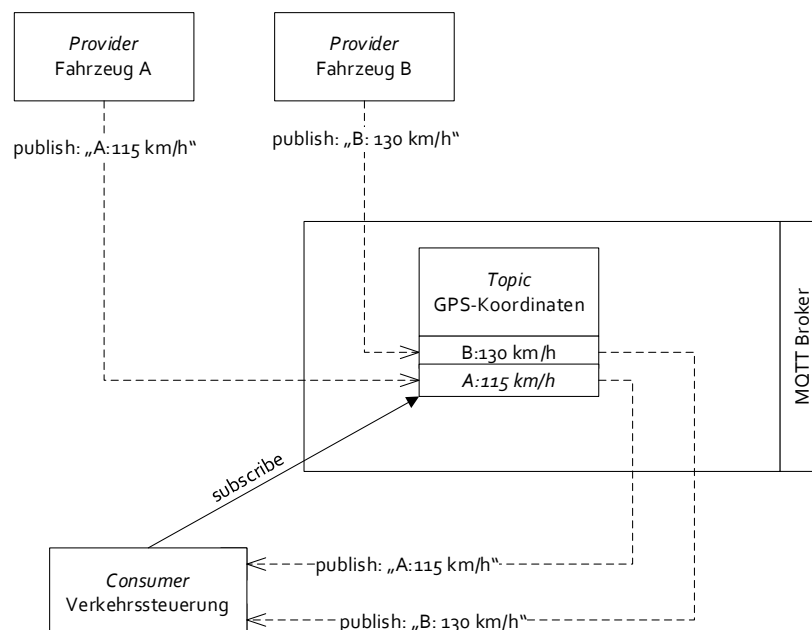


Abbildung 27: IoT Connected Car als Beispiel für Event-Driven-Integration

3.4. Technische Ebene: Klassifizierung von Integrationsverfahren nach Integrationsmethode

Segmentierung von Integrationsarchitekturen auf technischer Ebene

Der Begriff Integrationsverfahren beschreibt die Ausprägung der Systemkommunikation. Dies umfasst die technische Implementierung der System-zu-System-Kommunikation, bzw. Datenübertragung.

Abbildung 28 zeigt die Datenübertragung nach DIN 44302. Dieser erfolgt zwischen Datenstationen über einen gemeinsamen Übertragungsweg. Die Datenübertragung wird dabei von Datenübertragungseinrichtungen realisiert. [11]

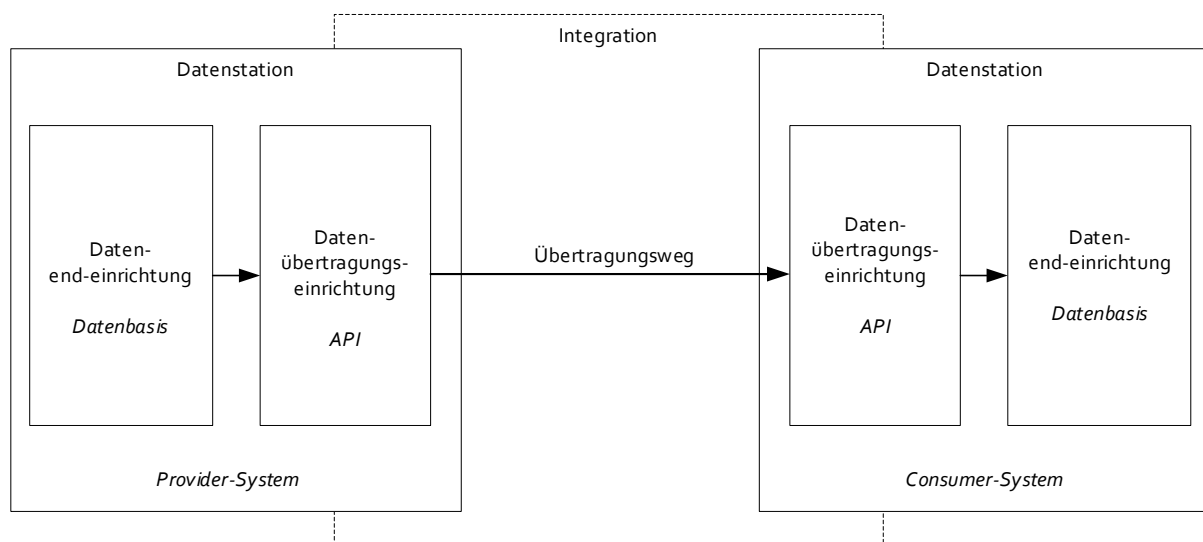


Abbildung 28: Datenübertragung nach DIN 44302
(Quelle: Eigene Darstellung in Anlehnung an DIN 44302 [61])

Im Kontext der Integration von Applikationssystemen entspricht die Datenstation dem Provider-System auf Sender- und dem Consumer-System auf empfangender Seite. Die Integration wird dabei zwischen zwei Übertragungseinrichtungen abgewickelt. Die Datenendeinrichtung beschreibt dabei die Datenbasis. Die Systemintegration ergibt sich aus dem Zusammenspiel dieser Komponenten, wobei die Übertragungseinrichtung sowohl systemnativ als auch nachträglich zum System hinzugefügt sein kann. [12]

Basierend auf dem oben beschriebenen Modell können Integrationsverfahren auf Grundlage von der jeweiligen Implementierung von Datenendeinrichtung und Datenübertragungseinrichtung gruppiert werden.

Betrachtet man beispielsweise die von Microsofts Cloud Service „Azure Logic Apps“ unterstützten Connectors, so sind gängige Integrationsmethoden unter Anderem File Transfer (FTP, Flatfile, XML), Web-APIs (SOAP, REST), Messages (RSS, SMTP, Queue), oder Datenbankverbindungen (ODBC, DB Link). [62]

Diese Integrationsmethoden unterscheiden sich in Bezug auf die Datenendeinrichtung und Datenübertragungseinrichtung im Wesentlichen im Format der Endeinrichtung, sowie der Kommunikationsart der Übertragungseinrichtung. Die Formate sind entweder *dateibasiert*, *webbasiert*, oder *tabellenbasiert*. Die Kommunikation erfolgt dabei analog zur Kategorisierung der Integrationsmuster entweder ausgehend vom Provider im als *Push*-Pattern oder ausgehend vom Consumer als *Pull*-Pattern. Zusätzlich dazu ist für die Bewertung der Integrationsmethode relevant, ob es die Übertragungseinrichtung inhärenter Teil des Systems ist (*intern*), oder nachträglich hinzugefügt, oder erweitert wurde (*extern*). Das entsprechende Klassifikationsschema ist in Abbildung 29 dargestellt.

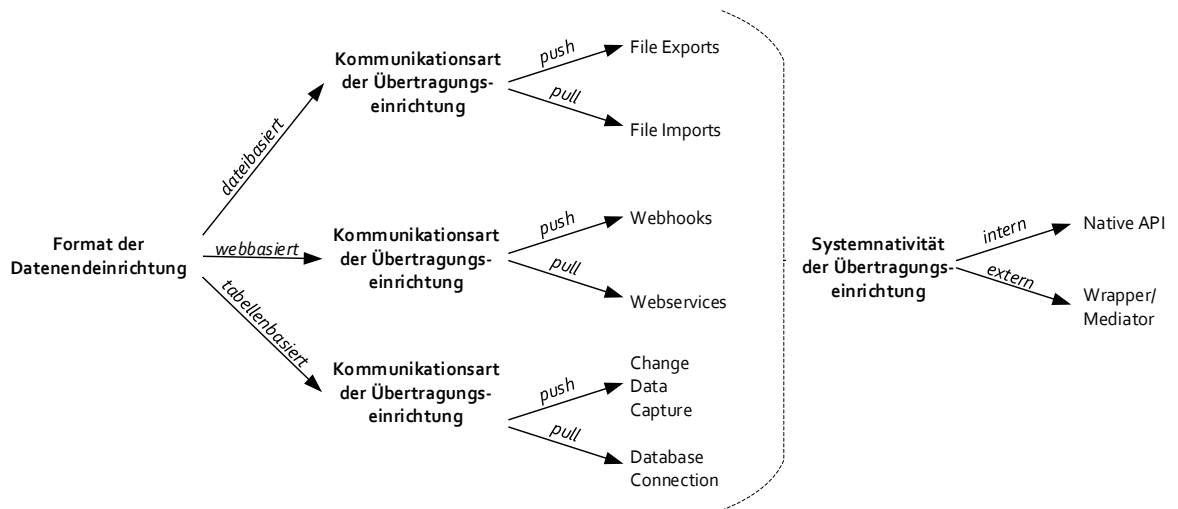


Abbildung 29: Klassifizierung von Integrationsmethoden

Abbildung 30 zeigt eine beispielhafte Klassierung von Integrationsmethoden. Bei dieser Klassifizierung wurden wie oben beschrieben nur gängige Integrationsmethoden im Kontext der Wirtschaftsinformatik beschrieben. Domänenspezifische Integrationsansätze, wie Feldbusse (KNX, MODBUS, CANBUS) oder Serielle Schnittstellen finden in diesem Modell keine Anwendung.

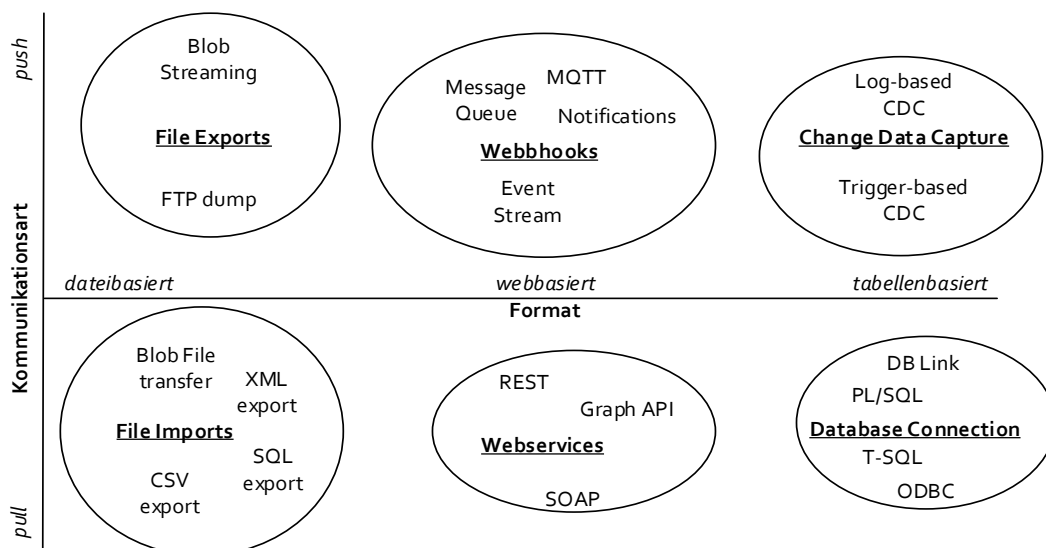


Abbildung 30: Beispielhafte Klassifizierung von Integrationsmethoden

Entsprechend der oben beschriebenen Überlegung lassen sich Integrationsverfahren wie in Abbildung 29 dargestellt in die Klassen **File-Exports**, **File-Imports**, **Webhooks**, **Webservices**, **Change-Data-Capture** und **Database-Connection** einordnen. Zusätzlich können Integrationsverfahren nach Nativität unterschieden werden, so dass sich **Native API** und **Wrapper/Mediator** gegenüberstehen.

3.4.1. File-Exports

Merkmale: Dateibasiertes Format, Push-Kommunikation

Unter der Methode *File-Exports* werden alle *dateibasierten* Integrationsverfahren zusammengefasst, bei denen die Kommunikationsart der Übertragungsrichtung im *Push-Verfahren* implementiert ist. Dabei werden die zu übertragenden Daten zunächst als Datei extrahiert und dann vom Provider-System in das Consumer-System übertragen.

Diese Methode ermöglicht den massenhaften Austausch von Daten in kurzer Zeit und findet vor allem Anwendung, wenn große Mengen von statischen Daten aus einem System extrahiert werden sollen. Ein Beispiel hierfür ist die Implementierung eines Snapshots zur späteren Datenanalyse, bei dem regelmäßig ein Massenexports von Daten aus einer SQL-Datenbank als XML-Formatdatei an einen FTP-Server gesendet wird. [63]

3.4.2. File-Imports

Merkmale: Dateibasiertes Format, Pull-Kommunikation

Als *File-Imports* werden alle *dateibasierten* Integrationsverfahren bezeichnet, bei denen die Kommunikationsart der Übertragungsrichtung im *Pull-Verfahren* implementiert ist. Bei dieser Methode werden die Daten vom Provider-System als Datei zur Verfügung gestellt und vom Consumer-System eingelesen.

File-Imports ermöglichen das schnelle Einlesen großer Datenmengen, welche zu einem festen Zeitpunkt zur Verfügung gestellt wurden. Ein Beispiel hierfür ist das Vorfüllen einer Datenbank über eine massenhafte Integration von Daten aus einer CSV-Formatdatei auf einem Blob-Storage-Server. [64]

3.4.3. Webhooks

Merkmale: Webbasiertes Format, Push-Kommunikation

Webbasierte Integrationsverfahren mit *Push-Kommunikation* werden als *Webhooks* bezeichnet. Dabei kontaktiert das Provider-System bei Bedarf das Consumer-System und informiert dieses über die Datenänderung.

Webhooks eignen sich für die zielgenaue und unmittelbare Übertragung kleiner Datenmengen, ohne dass das Consumer-System diese aktiv einholen muss. Ein Beispiel für Webhooks ist die Implementierung einer Event-Notification bei der das Provider-System automatisch eine Nachricht mit der erfolgten Datenänderung als JSON-Objekt über HTTP an das Consumer-System sendet. [65]

3.4.4. Webservices

Merkmale: Webbasiertes Format, Pull-Kommunikation

Unter der Kategorie *Webservices* werden alle webbasierten Integrationsverfahren zusammengefasst, bei denen Consumer-Systeme Daten vom Provider-System anfragen (*Pull-Kommunikation*). Dabei kontaktiert das Consumer-System den Provider und sendet proaktiv die gewünschten Anfragen zu Daten und Funktionalität und warten auf entsprechende Antwort. Diese Anfragen können dabei einmalig bei Bedarf (on Action), oder in regelmäßigen Zyklen (Polling) gestaltet werden. [66]

Über Webservices können Systeme schnell und leichtgewichtig eng miteinander Kommunizieren und Daten und Funktionalität flexibel miteinander teilen. So kann beispielsweise die Routenberechnung von Google Maps über die entsprechende REST-API nahtlos in eine Webapplikation integriert werden und in Echtzeit genutzt werden. [67]

3.4.5. Change-Data-Capture

Merkmale: Tabellenbasiertes Format, Push-Kommunikation

Change-Data-Capture (CDC) beschreibt Integrationsverfahren, bei denen Änderungen an Daten in *tabellenbasierten* Systemen vom Provider proaktiv kommuniziert werden (*Push-Pattern*). Der Datenaustausch kann dabei einerseits über die Datenbank-Logs, oder auch über Trigger auf Tabellen realisiert werden. Die Änderungen werden dabei automatisch erfasst und über Subscriber Views veröffentlicht.

CDC eignet sich besonders für die Integration von transaktionsorientierten Systemen, ist robust und gehört zur Standardfunktionalität professioneller Datenbankmanagementsysteme. Eine Abstraktion der Daten oberhalb der Tabellenebene erfolgt jedoch nicht. Als Beispiel können Änderungen des Lagerbestands im Warenwirtschaftssystem direkt in die Applikationsdatenbank des Onlineshops übertragen werden. [68]

3.4.6. Database-Connection

Merkmale: Tabellenbasiertes Format, Pull-Kommunikation

Die Kategorie *Database-Connection* umfasst alle tabellenbasierten Integrationen, bei denen Client-Applikationen aktiv auf die Provider-Systeme zugreifen (*Pull-Kommunikation*). Dabei stellt das Consumer-System über eine Schnittstelle eine Verbindung zu den in Tabellenstruktur abgelegten Daten des Providers.

Mit Database-Connections können große Datenmengen ausgetauscht, oder Operationen auf Datenbankebene ausgeführt werden. Typischerweise ist für die Datenübertragungseinrichtung ein spezifischer Treiber vonnöten, weshalb diese Art der Integration eine gewisse Infrastruktur voraussetzt und nicht sehr leichtgewichtig ist. Ein klassisches Beispiel ist die Implementierung einer Integration, bei der Daten des Providers über ODBC ausgelesen werden und durch den Consumer weiterverarbeitet werden. [69] Auch Interaktionen über prozedurale Befehle, beispielsweise in PL/SQL, oder T-SQL, sind im Rahmen einer Database-Connection möglich. [70]

3.4.7. Wrapper/Mediator

Merkmal: Systemnativität der Übertragungseinrichtung ist extern

Die Begriffe *Wrapper* und *Mediator* bezeichnen Integrationen, bei denen Integrationsmethoden nachträglich ertüchtigt oder erweitert wurden. Die Systemnativität der Übertragungseinrichtung ist somit nicht gegeben, bzw. systemextern.

API-Wrapper

Bei einer Wrapper-Architektur werden Systeme mit einer weiteren Komponente umschlossen. Ein Anwendungszweck für API-Wrapper sind Systeme, welche die benötigte Integrationsmethode nicht unterstützen und mit einem API-Wrapper ertüchtigt werden. Des Weiteren bietet ein API-Wrapper die Möglichkeit zur Reduzierung von Komplexität und Steigerung der Wertschöpfung. [71] APIs können hiermit in Bezug auf Schnittstellentechnologie und -design, oder auch Datenmodelle standardisiert und vereinfacht werden. Auch eine funktionelle Erweiterung ist mit der API-Wrapper-Methode möglich, ohne dass das zugrundeliegende System angepasst werden muss.

Wie in Abbildung 31 dargestellt, wird Die Schnittstelle mit der API-Fassade implementiert und logisch mit dem entsprechenden System verbunden. Consumer kommunizieren ausschließlich mit der API-Fassade. Nachdem der Consumer an Anfrage an die API-Fassade gestellt hat, interpretiert die Middleware diese Anfrage. Entsprechend der implementierten Logik greift die Middleware dann auf das Provider-System zu und führt die gewünschte Aktion aus und beantwortet mit der Rückantwort die ursprüngliche Anfrage des Consumers.

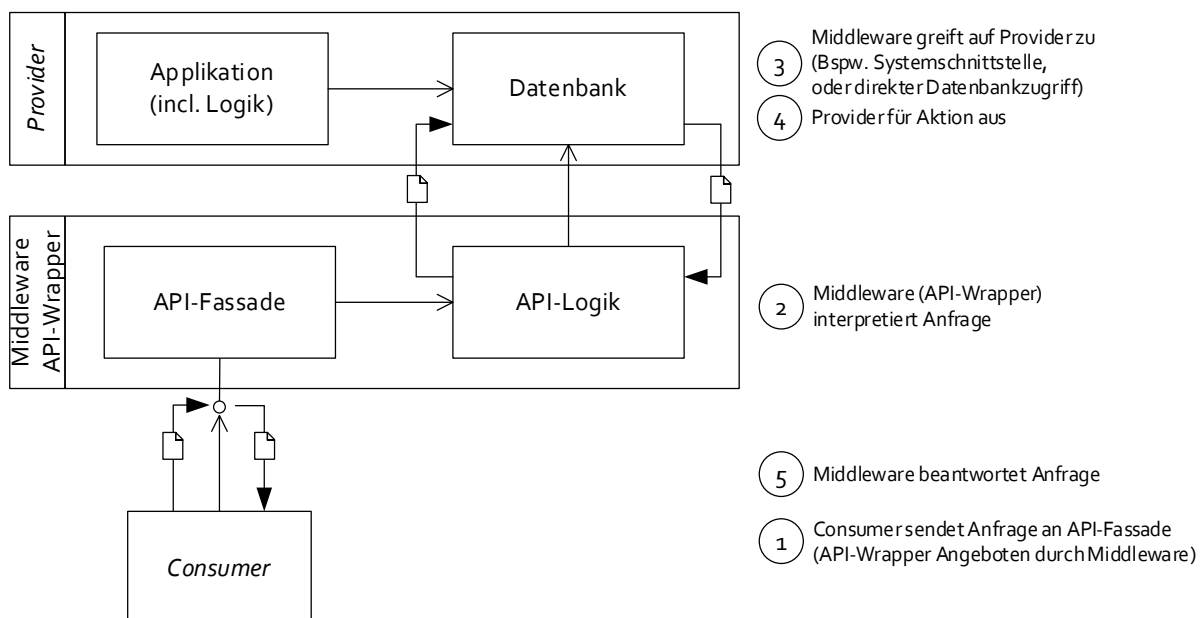


Abbildung 31 Schematische Darstellung eines API-Wrappers

API-Mediator

Bei einer Mediator-Architektur werden die APIs verschiedener Applikationen orchestriert und zu einer neuen API zusammengefasst. So können beispielsweise bestehende Services miteinander zu einem höherwertigen konsolidierten Service kombiniert werden. [72] Bei der Orchestrierung handelt es sich dann um eine eigene API mit eigener Logik, welcher die entsprechenden ange-bundenen APIs nutzt. [59]

Wie in Abbildung 32 dargestellt, kann für die Implementierung eines API-Mediators eine Middle-ware genutzt werden. Stellt ein Consumer eine Anfrage an den Mediator, so sendet die Middle-ware entsprechende Anfragen an die integrierten Provider. Je nach implementierter Logik kann dies parallel, oder aufeinander aufbauend unter Verarbeitung der jeweiligen Zwischenergebnisse gestaltet werden. Nachdem alle Provider die Anfragen beantwortet haben und die Middleware die Antworten bearbeitet hat, erhält der Consumer die Antwort auf seine Anfrage. Dies ist für den Consumer nicht sichtbar, da er lediglich mit dem Orchestrierungsservice kommuniziert.

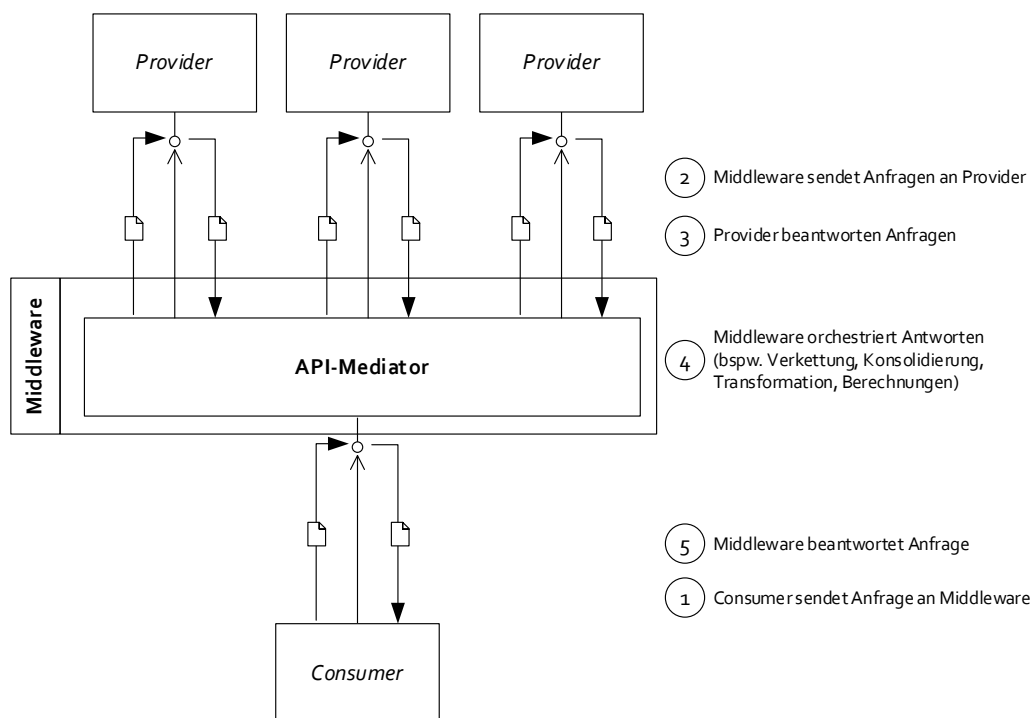


Abbildung 32: Beispielhafte Orchestrierung von Microservices in einer Middleware

3.5. Zusammenfassung: Definition der Taxonomie

Integrationsarchitekturen lassen sich nach der zugrundeliegenden Topologie, dem Paradigma, sowie der genutzten Methode in verschiedenen Klassen einordnen. Anhand dieser Taxonomie lässt sich jede beliebige Integrationsarchitektur entsprechend den Merkmalen ihrer Designartefakte analysieren und einem Architekturarchetypen zuordnen, welcher sich aus der Kombination der jeweiligen Klassifizierung auf den verschiedenen Granularitätsebenen ergibt. Tabelle 1 zeigt eine Übersicht dieser Klassifizierungsoptionen.

Topologie	Paradigma	Methode
Hub	Storage-Driven	File-Exports
Tree	Service-Driven	File-Imports
Bus	Event-Driven	Webhooks
Mesh		Webservices
		Change-Data-Capture
		Database-Connection

Tabelle 1: Übersicht der Klassifizierungsoptionen

3.5.1. Enterprise-Ebene: Klassifizierung nach Integrationstopologie

Die Enterprise-Ebene betrachtet Integrationslandschaften im größten Detailgrad und enthält die Gesamtheit aller IT-Systeme und -Anwendungen innerhalb einer Organisation. [6, Seite 2] Tabelle 2 zeigt die Klassifizierung dieser Integrationslandschaften nach Integrationstopologie.

Topologie (Enterprise-Ebene)		Kommunikationsflexibilität	
		<i>exklusiv</i>	<i>distributiv</i>
Kontrollstrategie	<i>zentral</i>	Hub	Bus
	<i>verteilt</i>	Tree	Mesh

Tabelle 2: Klassifizierung von Integrationslandschaften nach Integrationstopologie (Enterprise-Ebene)

Eine Klassifizierung von Integrationslandschaften anhand der Topologien erfolgt anhand von zentraler, bzw. verteilter Kontrollstrategie und exklusiver, bzw. distributiver Kommunikationsflexibilität. [7] Ausprägungen sind Integration-Hub, -Bus, -Tree und -Mesh. [8]

3.5.2. Applikationsebene: Klassifizierung nach Integrationsparadigma

Die Applikationsebene beschreibt die Integrationsarchitektur zwischen einzelnen Systemen, welche wiederum aus mehreren Komponenten bestehen können. [9, Seite 21] Tabelle 3 zeigt die Klassifizierung dieser Integrationsmustern nach Integrationsparadigma.

Paradigma (Applikationsebene)		Ausführung der Informationsintegration	
		<i>materialisiert</i>	<i>virtuell</i>
Initiierung des Informationsaustausches	<i>proaktiv (pull)</i>	Storage-Driven	Service-Driven
	<i>reaktiv (push)</i>	Event-Driven	

Tabelle 3: Klassifizierung von Integrationsmustern nach Integrationsparadigma (Applikationsebene)

Eine Klassifizierung der Integrationsmuster nach zugrundeliegenden Paradigmen erfolgt auf Grundlage von proaktiver, oder reaktiver Initialisierung des Informationsaustausches, sowie der materialisierten, bzw. virtuellen Ausführung der Informationsintegration. [10] Hieraus ergeben sich die Klassen Storage-Driven, Service-Driven- und Event-Driven-Integrations.

3.5.3. Technische Ebene: Klassifizierung nach Integrationsmethode

Die Technische Ebene erfasst die technische Implementierung einer bestimmten Komponente innerhalb der Systemlandschaft. [9, Seite 20–25] Tabelle 4 zeigt die Klassifizierung verschiedener Integrationsverfahren nach der genutzten Integrationsmethode.

Methode (Technische Ebene)		Format der Datenendeinrichtung		
		<i>dateibasiert</i>	<i>webbasiert</i>	<i>tabellenbasiert</i>
Kommunikationsart der Übertragungseinrichtung	<i>push</i>	File-Exports	Webhooks	Change-Data-Capture
	<i>pull</i>	File-Imports	Webservices	Database-Connection
Systemnativität der Übertragungseinrichtung	<i>intern</i>	Native API		
	<i>extern</i>	API-Wrapper/-Mediator		

Tabelle 4: Klassifizierung von Integrationsverfahren nach Integrationsmethode (Technische Ebene)

Eine Klassifizierung der Integrationsverfahren nach genutzten Methoden erfolgt auf Grundlage des Formats der Datenendeinrichtung (dateibasiert, webbasiert, bzw. tabellenbasiert) sowie Kommunikationsart der Übertragungseinrichtung (push. bzw. pull). [11] Hieraus ergeben sich die Integrationsmethoden File-Exports und -Imports, Webhooks und Webservices, sowie Change-Data-Capture und Database-Connection. Als weiteres Unterscheidungsmerkmal dient die Systemnativität der Übertragungseinrichtung, welche eine ergänzende Einordnung als Native API, bzw. Wrapper/Mediator-Integration erlaubt. [12]

4. Konzeption des GAP-Modells zur Analyse und Bewertung von Integrationsarchitekturen

Dieses Kapitel beschreibt die deduktive Konzeption eines Ansatzes zum modellgetriebenen Design von Integrationsarchitekturen und beantwortet die Forschungsfrage FF 3 (*Wie lassen sich Integrationsarchitekturen in Bezug auf die individuellen Anforderungen modellbasiert bewerten?*)

Der Autor konzipiert ein automatisierbares Mess- und Entscheidungsmodell für Softwarearchitektur. Dieses adressiert die in Kapitel 2 beschriebenen Herausforderungen bei Digitalisierung mit APIs und der massenhaften API-fizierung in historisch gewachsenen Systemlandschaften und fußt auf den in Kapitel 3 beschriebenen Ansätze zur Klassifizierung von Integrationsarchitekturen.

Dieses Modell wird in Folgendem als „GAP-Modell“ bezeichnet. „GAP“ steht als Akronym für „Good Architecture Practice“ und ist angelehnt an den GxP-Begriff aus dem Bereich Qualifizierter Systeme. [1, Seite 5] Neben einer konsequenten Strukturierung und Standardisierung anhand messbarer Kriterien impliziert das GAP-Modell gleichzeitig das Überwinden von Hindernissen hin zu einer guten Architektur im Sinne eines modernen Konnektivitätsökosystems.

Das GAP-Modell ist in das TOGAF Architekturrahmenwerk eingebunden und kann somit als Teil eines strukturierten iterativen Architekturplanungsprozesses auf beliebig großer Skala eingesetzt werden.

4.1. Anforderungen an ein Modell für Architekturentscheidungen

Eine zentrale Herausforderung bei der Digitalen Transformation besteht in dem zweckdienlichen Design von Systemintegrationslandschaften. Historisch gewachsene Systemlandschaften zeigen oftmals eine „Accidental Architecture“ ohne spezifische technologische und konzeptionelle Standards. Die Digitale Transformation erfordert jedoch moderne Konnektivitätsökosysteme mit guten Systemintegrationen.

Das Institut für deutsche Wirtschaft benennt in seiner aktuellen Studie „Digitalisierungsindex 2022“ fehlende Expertise und begrenzte Ressourcen als Hemmnis für die erfolgreiche Umsetzung der digitalen Transformation. [4, Seite 38] Um dieser begrenzten Kompetenzsouveränität zu begegnen und die digitale Transformation zu beschleunigen entwirft der Autor einen modellbasierten Ansatz für gut designte Integrationsarchitekturen. Im Fokus steht dabei die massenhafte API-fizierung, so dass das Modell sowohl für bestehende als auch für neu zu gestaltende Integrationsarchitekturen geeignet sein muss. Hieraus ergeben sich vier Kernanforderungen an das GAP-Modell, welche in Folgendem genauer beschrieben:

- Automatisierbarkeit für den massenhaften Einsatz
- Einfache Anwendung und Klarheit der Empfehlungen
- Generische Bewertungsmodell zur Bestandsanalyse
- Modellbasierte Unterstützung von Designentscheidungen

4.1.1. Automatisierbarkeit für den massenhaften Einsatz

Zielvorstellung ist ein automatisierbarer Ansatz, welcher den Prozess der Gestaltung von Integrationsarchitekturen strukturiert unterstützt. Hierbei soll bei möglichst geringem Ressourceneinsatz eine möglichst große Wirkleistung erzielt werden können. Idealerweise wird auf Grundlage einer einfachen Eingabe automatisch eine Entscheidungsvorlage für das Architekturdesign geliefert.

4.1.2. Einfache Anwendung und Klarheit der Empfehlungen

Für den massenhaften Einsatz bei gleichzeitig beschränkter Fachkräfteverfügbarkeit muss das Modell möglichst klar und einfach anwendbar sein. Die fachliche Kompetenz soll sich möglichst auf die Parametrisierung des Modells, sowie die Validierung und Umsetzung der generierten Designempfehlungen konzentrieren.

4.1.3. Generische Bewertungsmodell zur Bestandsanalyse

Obschon das zu definierende Bewertungsmodell für isolierte Architekturentscheidungen angewendet werden kann, so wird es so für den massenhaften Einsatz in einen größeren Kontext eingebunden. In erster Linie muss die Qualität existierender Integrationsarchitekturen bewertet werden können. So kann priorisiert werden, an welchen Stellen Handlungsbedarf entsteht. Der Fokus liegt dabei auf der Transformation historisch gewachsener Systemlandschaften, wobei gute Architekturentscheidungen selbstverständlich auch im Kontext neu entstehenden Umgebungen ohne Altsysteme anwendbar sind.

4.1.4. Modellbasierte Unterstützung von Designentscheidungen

Zusätzlich zur Bestandsanalyse muss auch aufgezeigt werden können, welche Designentscheidung für den konkreten Anwendungsfall am besten geeignet sind. Somit wird der Designprozess für Systemintegrationen durch die modellgetriebenen Designempfehlungen enorm vereinfacht.

Das Modell muss hierfür alle möglichen Lösungsoptionen identifizieren und dann anhand individueller Anforderungsprofile, sowie technischer, betrieblicher und organisatorischer Rahmenbedingungen kalibriert werden können, um möglichst universell einsetzbar zu sein.

4.2. Definition von Bewertungskriterien

Entsprechend der in Abschnitt 4.1 beschriebenen Anforderungen muss das GAP-Modell grundsätzlich automatisierbar und für den massenhaften Einsatz geeignet sein. Hierfür sind sowohl Einfachheit in der Anwendung des GAP-Modells, sowie Klarheit bei der daraus resultierenden Architekturempfehlungen entscheidend. Zudem muss das Modell so gestaltet sein, dass sowohl die Passgenauigkeit bestehender Integrationsarchitekturen als auch die Eignung potenziell zu implementierender Integrationsarchitekturen bewertet werden können.

Um den Anforderungen gerecht zu werden, muss das Bewertungsmodell Integrationsarchitekturen standardisiert anhand eines individuellen Anforderungsprofils bewerten können. Hierfür müssen Integrationsarchitekturen qualitativ betrachtet und eingeordnet werden. Die Dimensionen hierfür ergeben sich aus den in folgendem definierten Bewertungskriterien.

Ausgangspunkt für die Definition der Bewertungskriterien ist die ISO 27000, sowie der hierauf basierende BSI-Standard 200 zum IT-Grundschutz. Hierbei werden Informationen und die damit verbundenen Prozesse, Systeme und Netzwerke als kritischen Wirtschaftsgüter beschrieben [13, Seite 13]. Entsprechend ist die Betrachtung der Integrationsarchitektur im BSI-Standard 200-2 als Teil der Strukturanalyse von bei der Sicherheitsbewertung von Informationssystemen explizit vorgesehen. [73, Seite 82–85] Auf dieser Grundlage definiert der Autor folgende Bewertungskriterien, welche im weiteren Verlauf genauer beschrieben werden:

- Robustheit
- Integrität
- Skalierbarkeit
- Kontrolle
- Flexibilität
- Durchsatz
- Latenz

Die Bewertung erfolgt im Sinne der einfachen Anwendung auf einer vierstufigen Ordinalskala von *sehr schwach* (1), über *schwach* (2), *stark* (3), bis *sehr stark* (4). Die Abstände zwischen den Bewertungsausprägungen sind dabei gleich groß.

4.2.1. Robustheit

Robustheit beschreibt, in welchem Maßstab die integrierten Daten und Funktionalitäten verfügbar sind. [13, Seite 12]

Die Robustheit einer Integration lässt sich über die Ausfallzeiten der aktiven Systeme erfassen und ist je nach Anforderungsprofil in unterschiedlichem Maß akzeptabel. So können in einem Anwendungsfall kurze Ausfallzeiten akzeptiert werden, wohingegen andere Systemintegrationen nicht ausfallen dürfen, da die integrierten Daten und Funktionalitäten für Durchführung geschäftsentscheidender Aktivitäten jederzeit benötigt werden. [73, Seite 24–25]

Ein Hohes Maß an Robustheit wird im GAP-Modell besser bewertet als ein geringes.

Für die Robustheit einer Integrationsarchitektur ist entscheidend, wie gut das Konnektivitätsökosystem auf den Ausfall einzelner Komponenten reagiert. In Bezug auf Topologien ist hierbei insbesondere relevant, ob es einen Single-Point-of-Failure gibt. Im Kontext der Integrationsparadigmen wird insbesondere betrachtet, ob Consumer im Falle eines Ausfalls von Providern noch arbeitsfähig sind.

4.2.2. Integrität

Integrität beschreibt den Grad der Richtigkeit und Vollständigkeit Daten. [13, Seite 5]

Die Integrität einer Integration ist an der Quote der fehlerhaften Informationen in den jeweiligen Systemen erkennbar. So sind beispielsweise im normalen Umfeld kleine Fehler in der Datenbasis tolerierbar, wenn diese die Aufgabenerfüllung nicht beeinträchtigen, wohingegen Prozesskritische Informationen immer korrekt und aktuell sein müssen. [73, Seite 24–25]

Hohe Integrität wird im GAP-Modell besser bewertet als geringe Integrität.

Integrationen mit hoher Integrität zeichnen sich dadurch aus, dass Informationen im Konnektivitätsökosystem zuverlässig synchronisiert werden, ohne dass es zu Konflikten und Uneinigkeiten kommt. Eine hohe Integrität liegt dann vor, wenn die verfügbaren Informationen stets unverfälscht, authentisch und unbestreitbar sind. [73, Seite 14]

4.2.3. Skalierbarkeit

Skalierbarkeit beschreibt die Fähigkeit zur Anpassung von Systemen an einen veränderten Leistungsbedarf.

Skalierbare Integrationsarchitekturen zeichnen sich dadurch aus, dass für Änderungen an der Zusammensetzung des Konnektivitätsökosystems vorhandene Systeme nicht, oder nur kurzzeitig außer Betrieb genommen werden müssen und beliebig viele Akteure zum Ökosystem hinzugefügt werden können, ohne dass dies die bereits bestehenden Integrationen beeinträchtigt. [74, Seite 14]

Gute Skalierbarkeit wird im GAP-Modell besser bewertet als schlechte Skalierbarkeit.

In skalierbaren Integrationsarchitekturen lassen sich Systeme massenhaft zum Konnektivitätsökosystem hinzufügen und entfernen, ohne dass dies ungewünschte Auswirkungen auf die Funktionalität der Systeme hat. Es muss also möglich sein, zusätzliche Integrationen zu (de-)aktivieren, ohne dass es hierbei zu einer relevanten Unterbrechung des Betriebs kommt. [74, Seite 14]

4.2.4. Kontrolle

Kontrolle beschreibt die Verfügbarkeit von Informationen zum Management von Risiken, sowie die Fähigkeit diese zu beherrschen. [13, Seite 3–4]

Ein hohes Maß an Kontrolle wird im GAP-Modell besser bewertet als geringe Kontrolle.

Für die Integrationsarchitektur bedeutet Kontrolle, dass sie Komplexität der Systemlandschaft und der einzelnen Integrationen gut beherrschbar ist. So können Relevante Gefährdungen des Konnektivitätsökosystems strukturiert abgewehrt werden, da eine Einschätzung des tatsächlichen Lagebildes auf Grundlage der verfügbaren Informationen jederzeit möglich ist. [75, Seite 22–27]

4.2.5. Flexibilität

Flexibilität beschreibt die Fähigkeit auf Veränderungen zu reagieren und entsprechende Anpassungen umzusetzen (Resilienz). [76, Seite 14]

Hohe Flexibilität wird im GAP-Modell besser bewertet als geringe Flexibilität.

In flexiblen Integrationsarchitekturen lassen sich bestehende Strukturen einfach anpassen. Veränderungen können so jederzeit adressiert werden, ohne dass die operative Integrität des Konnektivitätsökosystems gefährdet wird. So werden Störungen und Unterbrechungen durch funktionale Änderungen vermieden. Flexibilität ist somit ein grundlegender Faktor für die Betriebskontinuität und sichert die Verfügbarkeit von Daten und Funktionalität im Normalbetrieb. [75, Seite 33–34]

4.2.6. Durchsatz

Durchsatz beschreibt das Leistungsvermögen der Kommunikation zwischen integrierten Systemen und ist definiert durch die Menge an Daten, welche in einer bestimmten Zeit übertragen werden, kann. [77]

Hoher Durchsatz wird im GAP-Modell besser bewertet als niedriger Durchsatz.

Der Durchsatz einer Integration lässt sich mit der Größe des durchschnittlich übertragenen Datenvolumens messen. Bei Integrationen mit geringem Durchsatz, kann nur eine stark begrenzte Menge an Informationen synchronisiert werden, da es ansonsten zu einer Überlastung kommt. Im Gegensatz dazu können bei hohem Durchsatz große Datenmengen innerhalb eines kleinen Zeitfensters übertragen werden. [78]

4.2.7. Latenz

Latenz beschreibt die Wartezeit zwischen der Aktualisierung einer Information und der tatsächlichen Datenübertragung. [77]

Eine geringe Latenz wird im GAP-Modell besser bewertet als eine hohe Latenz.

Eine niedrige Latenz bedeutet, dass Informationen unmittelbar übertragen werden und die Integration somit sehr reaktionsschnell abläuft. Verzögerungen bei der Benutzung der integrierten Systeme werden hierbei durch unverzügliches Senden und Empfangen minimiert. [78] Bei geringer Latenz sind die verarbeiteten Informationen immer aktuell und die zeitliche Lücke zwischen Informationen in Quell- und Zielsystem ist sehr klein.

4.3. Bewertung von Integrationsarchitekturen

In diesem Abschnitt beschreibt der Autor das GAP-Modell. Ausgehend von den in Abschnitt 4.1 beschriebenen Anforderungen wird ein generisches Bewertungsmodell für Integrationsarchitekturen definiert, welches an individuelle Gegebenheiten angepasst werden kann. Die Bewertung der Architekturen erfolgt dabei auf Grundlage der in Abschnitt 4.2 definierten Bewertungskriterien *Robustheit*, *Integrität*, *Skalierbarkeit*, *Kontrolle*, *Flexibilität*, *Durchsatz* und *Latenz*.

Abbildung 33 zeigt schematisch das in Folgendem beschriebene Vorgehen bei der Bewertung von Integrationsarchitekturen unter Anwendung des GAP-Modells.

Ausgangspunkt ist ein Individuelles Anforderungsprofil, welches sich aus einer anwendungsfall-spezifischen Gewichtung der Bewertungskriterien ergibt (*Konfiguration*). Dieses Anforderungsprofil ist die Grundlage zur Bewertung bestehender Architekturen und Identifikation der besten Alternative im Optionsraum. Dieser Optionsraum umfasst alle möglichen Architekturalternativen und bildet diese als Architekturarchetypen ab. Die Zuordnung von Architekturen zu Archetypen erfolgt dabei auf Grundlage der in Kapitel 3 definierten Taxonomie zur Klassifikation von Integrationsarchitekturen. Ein Archetyp ergibt sich aus der Kombination von Integrationstopologie, -paradigma und -methode.

Zur automatischen Berechnung der Bewertungsergebnisse wird zunächst die vergleichende Bewertung der Integrationsarchitekturen zugrunde gelegt (*Parametrisierung*). Die Bewertung eines Archetyps ergibt sich aus der Kombination der Wertzahlen der zugeordneten Architekturartefakte. Diese Bewertungen werden nun in Bezug zu den Gewichteten Kriterien des individuellen Anforderungsprofils gesetzt. Im Ergebnis unterstützt das GAP-Modell die qualifizierte Architekturentscheidung, indem passende Architekturarchetypen vorgeschlagen werden.

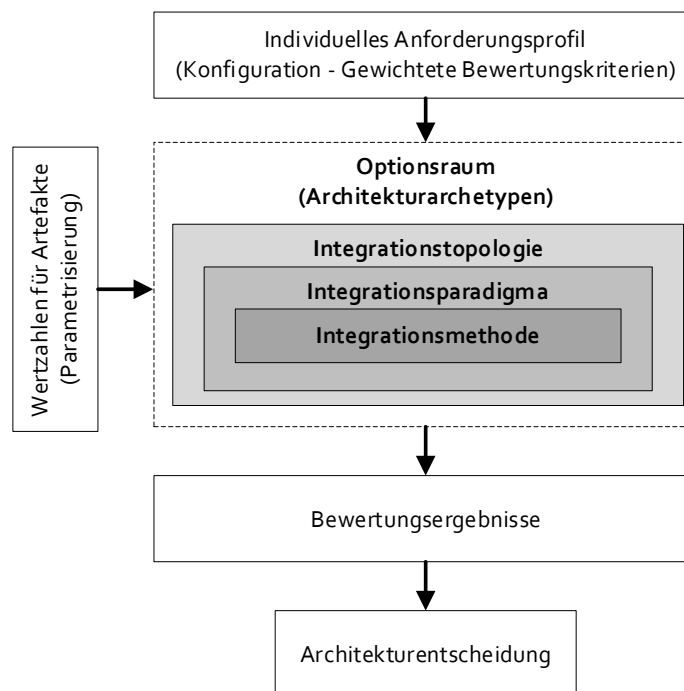


Abbildung 33: Vorgehen bei der Bewertung von Integrationsarchitekturen unter Anwendung des GAP-Modells

4.3.1. Konfiguration des individuellen Anforderungsprofils durch Gewichtung von Bewertungskriterien

Der erste Schritt bei der Anwendung des GAP-Modells besteht in der Definition des individuellen Anforderungsprofils. Hierfür werden die in Abschnitt 4.2 beschriebenen Bewertungskriterien anwendungsfallspezifisch gewichtet. Die Gewichtung der einzelnen Kriterien erfolgt über eine Gewichtungsmatrix. Dies ist Tabelle 5 beispielhaft dargestellt.

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Gewichtung	
Robustheit	-	2	3	3	2	2	3	15	17%
Integrität	2	-	3	2	3	3	3	16	19%
Skalierbarkeit	1	1	-	1	1	2	2	8	9%
Kontrolle	1	2	3	-	3	3	3	15	17%
Flexibilität	2	2	3	1	-	1	1	10	12%
Durchsatz	2	2	2	1	3	-	2	12	14%
Latenz	1	1	2	1	3	2	-	10	12%
Skala: 1 = unwichtiger 2 = gleich wichtig 3 = wichtiger								86	100%

Tabelle 5: Beispielhafte Gewichtungsmatrix

Die Gewichtung der Bewertungskriterien erfolgt durch direkten Vergleich der Wichtigkeit jedes Bewertungskriteriums mit den übrigen Kriterien. Für die abgestufte Gewichtungsberechnung wird eine gespiegelte Ordinalskala mit konstanten Abständen genutzt. Hierbei wählt der Autor im Sinne der einfachen Anwendung eine dreistufige Skala von 1 (*unwichtiger*) über 2 (*gleich wichtig*), bis 3 (*wichtiger*). Die Abstände zwischen den Bewertungsausprägungen sind gleich groß, wobei 1 (*unwichtiger*) und 3 (*wichtiger*) die Extrempunkte bilden und 2 (*gleich wichtig*) den Nullpunkt beschreibt.

Die Gewichtungsmatrix enthält für jedes Kriterium jeweils eine Spalte und analog eine Zeile. Nun wird für jede Zeile bewertet, ob das in der Spalte aufgeführte Kriterium eine kleinere, gleiche, oder größere Bedeutung als das Kriterium in der Zeile hat. Durch Aufsummierung der ermittelten Bewertungspunkte pro Zeilen wird abschließend die absolute Gewichtung ermittelt. Die relative Gewichtung ergibt sich aus der Division der Wertpunkte durch die Summe aller Wertpunkte. [79] Aufgrund des Symmetrischen Aufbaus der Matrix ist der Abstand zwischen den jeweiligen Paaren unter und oberhalb der Matrix-Mitte immer gleich. Wird also zum Beispiel *Robustheit* im Vergleich zu *Latenz* als 3 (*wichtiger*) bewertet, so muss analog *Latenz* im Vergleich zu *Robustheit* als 1 (*unwichtiger*) bewertet werden.

Ausgehend von der oben beschriebenen Gewichtungsmatrix lässt sich ein Anforderungsprofil für die Integrationsarchitektur definieren. Hierfür muss für jedes Kriterium ein Zielwert ermittelt werden. Wie in Abschnitt 4.3 beschrieben, erfolgt die Bewertung auf einer vierstufigen Ordinalskala von *sehr schwach* (1), über *schwach* (2), *stark* (3), bis *sehr stark* (4). Der Zielwert wird anhand der Spreizung zwischen theoretischem Maximal- und Minimalwert ermittelt.

Die Berechnung der Zielwerte ist in Abbildung 34 dargestellt.

$$\begin{aligned}
 rel_{min} &= (criteria_{count} - 1) * criteria_{min} \\
 rel_{max} &= rel_{min} * criteria_{max} \\
 target &\approx \frac{(rel - rel_{min})}{rel_{step}} * target_{max}
 \end{aligned}$$

Abbildung 34: Berechnung der Ziel-Bewertung des Anforderungsprofils

Der Minimalwert (rel_{min}) entspricht einer Bewertung aller Kriterien mit dem geringsten Wert. Dies entspricht bei der Mindestbewertung von 1 der Anzahl aller Bewertungskriterien ($criteria_{count}$) minus 1 multipliziert mit dem Minimalwert ($criteria_{min}$) auf der Ordinalskala der Gewichtung. Für den Maximalwert (rel_{max}) kann dieser Wert mit dem Maximalwert der Gewichtungsskala ($criteria_{max}$) multipliziert werden. Die Spreizung (rel_{step}) ergibt sich aus der Differenz von rel_{min} und rel_{max} . Zur Transformation der relativen Gewichtung zur Zielbewertung des Anforderungsprofils wird jeweils der Wert für die relative Gewichtung (rel) um rel_{min} subtrahiert, mit rel_{step} dividiert und mit der Maximalbewertung für die Zielbewertung ($target_{max}$) multipliziert. Dieser Wert wird dann entsprechend der Ordinalskala für die Zielbewertung auf eine ganze Zahl gerundet. Abbildung 35 zeigt ein beispielhaftes Anforderungsprofil.

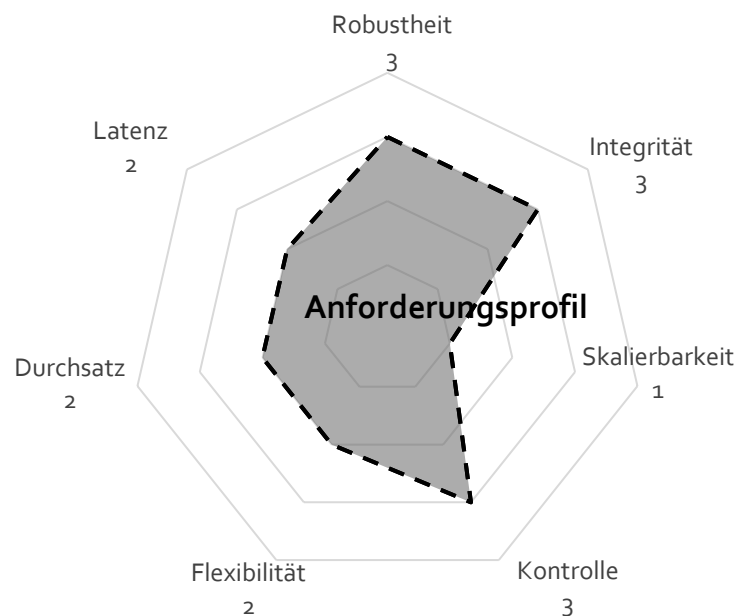


Abbildung 35: Beispielhaftes Anforderungsprofil

Das beschriebene Bewertungsmodell wurde vom Autor bereits in der Vergangenheit für andere Anwendungsfälle in ähnlicher Weise genutzt und validiert. So wurde beispielsweise 2020 nach diesem Vorgehensmodell in den Beschaffungsprozess einer Unternehmensweiten API-Management-Lösung in einem DAX-Konzern integriert und validiert. [46, Seite 23–33] Die Ergebnisse dieses anforderungsbasierte Bewertungsmodells konnten mit anderen Bewertungsansätzen, wie beispielsweise dem Gartner Magic Quadrant, bestätigt werden. [80] Letztendlich wurde die Auswahl und Beschaffungsentscheidung entsprechend der Ergebnisse der modellgetriebenen Bewertung getroffen.

4.3.2. Parametrisierung des Bewertungsmodells durch Bewertung von Designartefakten

Das GAP-Modell misst auf Grundlage der individuellen Anforderungen die qualitative Eignung bestimmter Integrationsstrukturen. Grundlage für diese Messung ist eine vergleichende Bewertung der verschiedenen Designartefakte, welche gemeinsam das Integrationsdesign bilden. Die Designartefakte werden dafür anhand der in Kapitel 3 ausgearbeiteten Taxonomie auf den in Abschnitt 2.3 beschriebenen Ebenen der IT-Architektur eingeordnet.

Die Bewertung jedes Aspektes erfolgt qualitativ auf einer Ordinalskala von *sehr schwach* (1), über *schwach* (2), *stark* (3), bis *sehr stark* (4). Die Abstände zwischen den Bewertungsausprägungen sind dabei gleich groß. Da das Ziel die Identifikation der für den Anwendungsfall am besten geeigneten Lösung ist, wird die Bewertung der jeweiligen Integrationstopologien -paradigmen und -methoden vergleichend durchgeführt.

Die Bewertung der Designartefakte erfolgt auf den Dimensionen, welche sich aus den in Abschnitt 4.2 beschriebenen Bewertungskriterien ergeben. Wie in Tabelle 6 dargestellt finden einige der Bewertungskriterien auf allen Ebenen Anwendung, wohingegen andere nur für ausgewählten Granularitätsebenen relevant sind.

Kriterium	Topologie	Paradigma	Methode
Robustheit	■	■	■
Integrität	■	■	■
Skalierbarkeit	■	□	□
Kontrolle	■	■	■
Flexibilität	■	■	■
Durchsatz	□	■	■
Latenz	□	■	■

Tabelle 6: Anwendungsbereiche der Bewertungskriterien

Für *Robustheit*, *Integrität*, *Kontrolle* und *Flexibilität* ist sowohl der konzeptionelle Aufbau der Integrationen, sowie die technische Implementierung entscheidend, weshalb diese Kriterien für die Bewertung auf allen drei Ebenen (Topologie, Paradigma und Methode) angewendet werden.

Skalierbarkeit ist vor allem getrieben durch den konzeptionellen Aufbau der Integrationslandschaft und findet daher nur bei der Bewertung der Topologie Anwendung.

Durchsatz und *Latenz* sind abhängig von der tatsächlichen Implementierung der direkten Systemintegration, weshalb dieser Punkte nur bei der Bewertung von Paradigma und Methode angewendet werden.

Tabelle 7 zeigt die Parametrisierung, welche in Kapitel 5 beschrieben ist und für die Validierung in Kapitel 6 verwendet wurde. Diese Parametrisierung liefert die initialen Wertzahlen für die jeweiligen Designartefakte für alle Dimensionen des Bewertungsmodells.

		Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz
Topologie	Hub	1	4	2	4	3	-	-
	Bus	2	2	3	3	4	-	-
	Tree	3	3	4	3	1	-	-
	Mesh	4	1	3	2	4	-	-
Paradigma	Storage-Driven	4	2	-	3	1	4	1
	Service-Driven	1	4	-	3	4	1	4
	Event-Driven	2	3	-	1	4	3	4
Methode	File-Exports	4	3	-	2	3	4	2
	File-Imports	4	1	-	2	3	4	1
	Webhooks	1	4	-	4	4	2	4
	Webservices	2	4	-	4	4	2	4
	Change-Data-Capture	3	3	-	3	2	3	4
	Database-Connection	2	4	-	2	2	3	4
	Wrapper / Mediator	1	-	-	-	-	-	-

Skala: 1 = sehr schwach 2 = schwach 3 = stark 4 = sehr stark

Tabelle 7: Initiale Parametrisierung des GAP-Modells (siehe Kapitel 6)

Wie in Kapitel 7 beschrieben, kann das GAP-Modell über eine Verfeinerung und Adjustierung der Parametrisierung potenziell weiter verbessert werden.

4.3.3. Aufbau des Optionsraums durch Kombination klassifizierter Designartefakte

Grundsätzlich definiert sich der Optionsraum als Menge aller möglichen Archetypen von Integrationsarchitekturen. Ein Architekturarchetyp ergibt sich aus der Kombination einer bestimmten Integrationstopologie mit einem bestimmten Paradigma und einer bestimmten Methode. Somit werden alle möglichen Kombinationen von Designartefakten auf den drei Granularitätsebenen der IT-Architektur abgebildet. Abbildung 36 zeigt einen beispielhaften Architekturarchetyp.

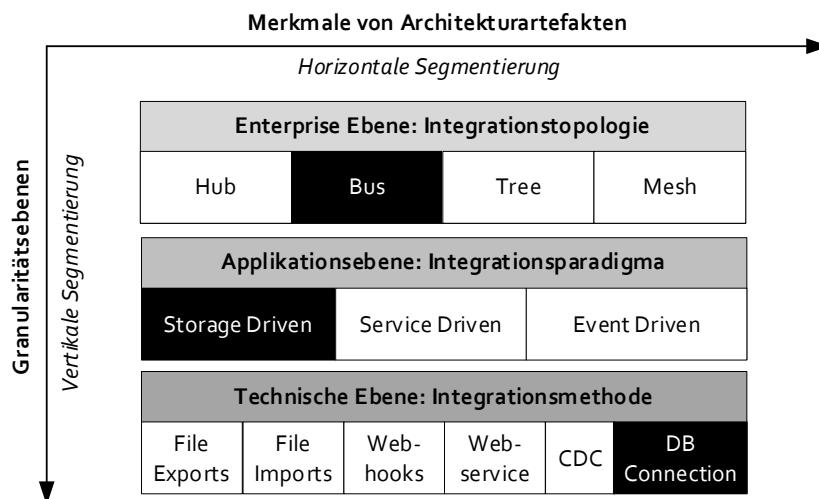


Abbildung 36: Architekturarchetyp „Bus-Topologie nach Storage-Driven-Paradigma mit Database-Connection“

Der Optionsraum kann aufgrund technischer, betrieblicher, oder organisatorischer Rahmenbedingungen eingeschränkt werden. Besteht beispielsweise der Zwang zu einem Integration-Bus, so werden alle anderen Topologien aus dem Optionsraum ausgeschlossen.

4.3.4. Bewertung der Architekturarchetypen im Optionsraum durch Anwendung des Bewertungsmodells

Für die Bewertung bestehender Integrationsarchitekturen müssen diese zunächst ein Archetyp zugeordnet werden. Dies erfolgt anhand einer Analyse der Merkmale der tatsächlich implementierten Systemintegration. Die identifizierbaren Strukturen schließen sich dabei nicht gänzlich aus, sondern können in einer tatsächlichen Implementierung miteinander kombiniert werden. [10, Seite 9–10] Daher muss eine bestehende zu bewertende Architektur unter Umständen segmentiert und separat betrachtet werden. Für neu zu implementierende Integrationen entfällt dieser Schritt.

Eine Bewertung von Integrationsarchitekturen erfolgt dann anhand einer gewichteten Bewertung der einzelnen Architekturarchetypen. Diese ergibt sich aus einer Kombination der Einzelbewertungen der jeweiligen Architekturklassen auf den verschiedenen Granularitätsebenen. Die hierfür benötigte initiale Parametrisierung des Bewertungsmodells ist in Kapitel 5 ausführlich beschrieben.

Tabelle 8 zeigt die Bewertungsmatrix für Designartefakte auf den drei Ebenen der IT-Architektur als Grundlage für die Bewertung der Integrationsarchitekturen. Für einzelne Methoden kann die Option Wrapper/Mediator gewählt werden, wodurch die Bewertung für Robustheit entsprechend angepasst wird (siehe Bewertung von Integrationsmethoden in Abschnitt 5.3)

Für jeden Archetypen berechnet sich die für jede Dimension die Bewertung als Durchschnitt (arithmetisches Mittel) der Bewertungen für die jeweilige Topologie, das Paradigma sowie die Methode. Im Beispiel ergibt sich für Robustheit eine Bewertung für eine Integration in Bus-Topologie (*schwach 2*) nach Storage-Driven-Paradigma (*sehr stark 4*) mit Database-Connection (*schwach 2*) ein Wert von *stark (3)*.

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Wrapper/Mediator
Hub	1	4	2	4	3	-	-	-
Bus	2	2	3	3	4	-	-	-
Tree	3	3	4	3	1	-	-	-
Mesh	4	1	3	2	4	-	-	-
Storage-Driven	4	2	-	3	1	4	1	-
Service-Driven	1	4	-	3	4	1	4	-
Event-Driven	2	3	-	1	4	3	4	-
File-Exports	1	3	-	2	3	4	2	
File-Imports	4	1	-	2	3	4	1	
Webhooks	1	4	-	4	4	2	4	x
Webservices	2	4	-	4	4	2	4	
Change-Data-Capture	1	3	-	3	2	3	4	x
Database-Connection	2	4	-	2	2	3	4	

Skala: 1 = sehr schwach 2 = schwach 3 = stark 4 = sehr stark

Tabelle 8: Bewertungsmatrix für Designartefakte auf den drei Ebenen der IT-Architektur

Der Autor wählt an dieser Stelle zunächst diese einfache Berechnungsmethode unter der Annahme, dass dies bereits eine aussagekräftige Bewertung der jeweiligen Architekturarchetypen erlaubt. Möglichkeiten zur weiteren Verbesserung des GAP-Modells, bspw. durch eine elaboriertere Ermittlung der kombinierten Bewertung werden im Ausblick in Kapitel 7 diskutiert.

4.3.5. Modellbasierte Architekturentscheidungen durch Auswertung der Bewertungsergebnisse

Abschließend werden durch den Abgleich von Zielbewertung und Bewertungen im Optionsraums geeignete Integrationsarchitekturen identifiziert. Alle Alternativen im Optionsraum können mit den Gewichteten Bewertungskriterien in verschiedenen Dimensionen mit diesem Anforderungsprofil abgeglichen werden

Als geeignet gelten alle Archetypen, bei denen allen Kriterien mindestens der Zielbewertung entsprechen, oder besser bewertet wurden. Können aufgrund des Anwendungsprofil keine passenden Architekturen identifiziert werden, so empfiehlt das GAP-Modell auch Architekturen, welche die Anforderungen knapp verfehlen.

Tabelle 9 zeigt alle Integrationsarchitekturen, die dem beispielhaft definierten Anforderungsprofil gerecht werden. Im dargestellten Beispiel wird nach einer ausgewogenen Lösung gesucht, bei der Robustheit, Integrität und Kontrolle eine erhöhte Wichtigkeit zugestanden werden. Eine passende Integrationsarchitektur muss dabei mindestens die in der Zeile „Ziel-Bewertung“ ausgezeichnete Zielmarke erreichen. In den Darunterliegenden Zeilen sind die Wertzahlen für die als passend identifizierten Architekturarchetypen aufgeführt.

Topologie	Paradigma	Methode	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Ziel-Bewertung			3	3	1	3	2	2	2	
Tree	Storage-Driven	Webservices	3	3	4	3	2	3	3	7,34
Bus	Storage-Driven	Webservices	3	3	3	3	3	3	3	7,23
Tree	Storage-Driven	Webhooks	3	3	4	3	2	3	3	7,16
Tree	Storage-Driven	File-Exports	4	3	4	3	2	4	2	7,03
Bus	Storage-Driven	Database-Connection	3	3	3	3	2	4	3	6,77
Tree	Service-Driven	File Imports	3	3	4	3	3	3	3	6,74

Skala: 1 = sehr schwach 2 = schwach 3 = stark 4 = sehr stark

Tabelle 9: Geeignete Integrationsarchitekturen

Ergänzt werden die Bewertungsergebnisse durch eine Gesamtbewertung. Hierbei wird für die weitere Differenzierung der Ergebnisse zusätzlich zu den bewerteten Anforderungskategorien für jeden Archetypen eine Gesamtbewertung berechnet. Diese ergibt sich für jeden Archetypen aus der Summe der Durchschnittswerte der Bewertungen aller Kategorien multipliziert mit der jeweiligen relativen Gewichtung pro Granularitätsebenen. [79]

Abbildung 37 zeigt beispielhaft die Visualisierung der Empfehlungen des GAP-Modells für passende Integrationsarchitekturen. Im betrachteten Beispiel wird das physische Kopieren (Storage-Driven) von Daten über Webservices direkt von einem System ins andere (Tree) empfohlen.

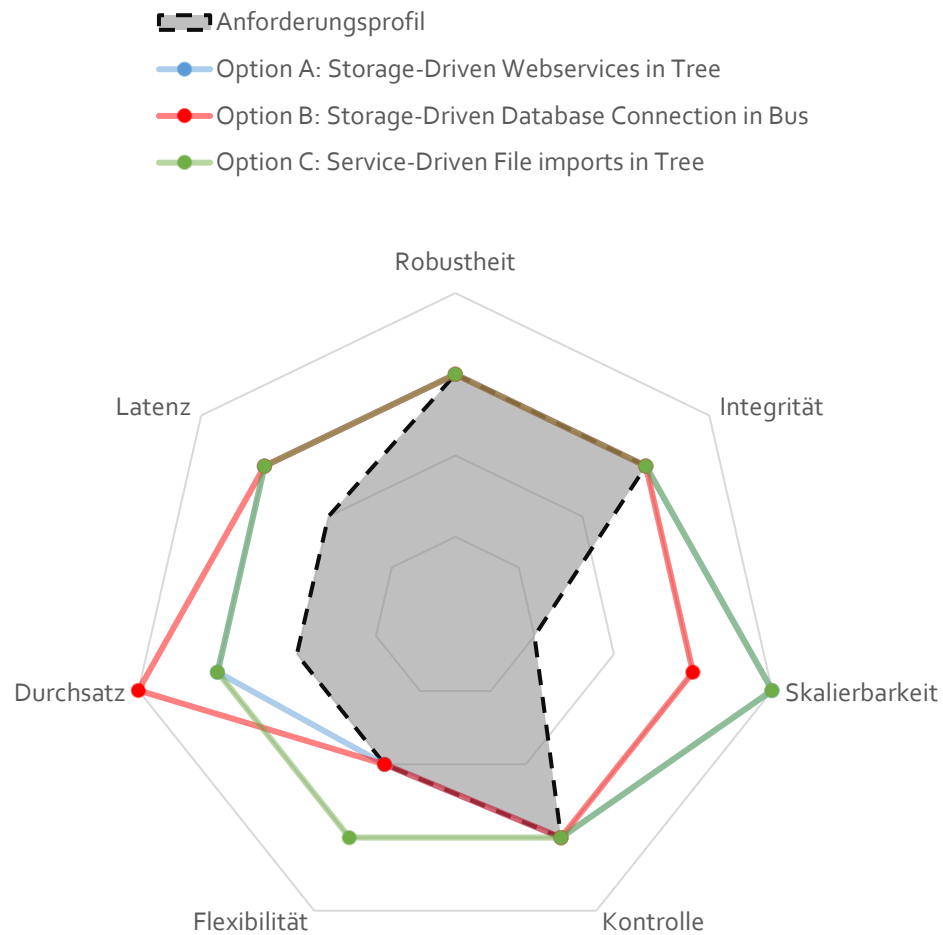


Abbildung 37: Beispielhafte Visualisierung der GAP-Modell Empfehlung

4.4. Einordnung in das TOGAF Architekturrahmenwerk

Für den massenhaften Einsatz benötigt das GAP-Modell solide Strukturen und Prozesse in deren Kontext es auf großer Skala eingesetzt werden kann. Das in Kapitel 2 beschriebene The Open Group Architecture Framework (TOGAF) bietet hierfür einen robusten Ordnungsrahmen.

Abbildung 38 zeigt das Konzipierte GAP-Modell und seine Einbettung in das TOGAF Rahmenwerk. Das GAP-Modell ist in dabei eine konkrete Implementierung der TOGAF Architecture Alternatives Method im Kontext der iterativen Architecture Development Method.

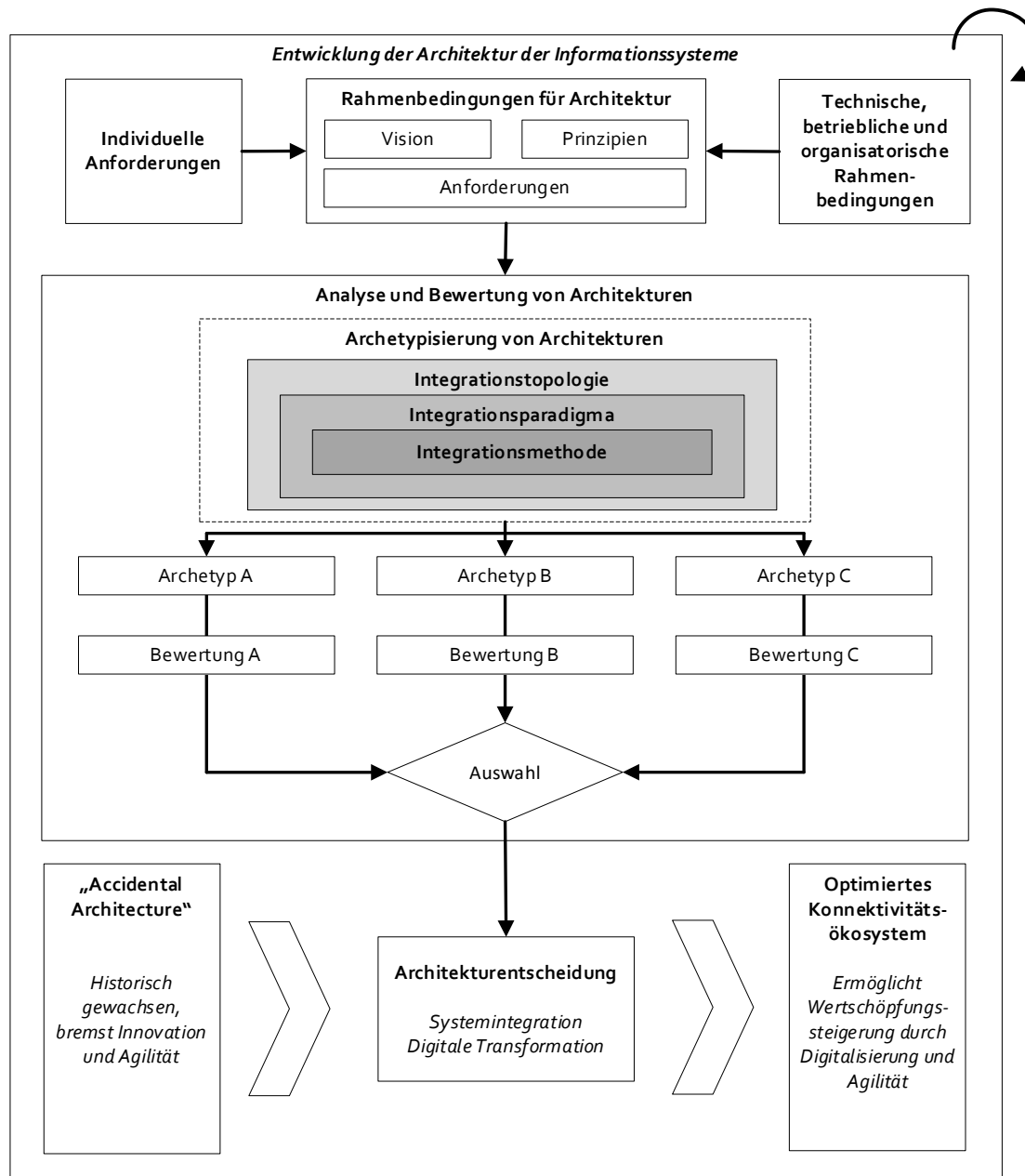


Abbildung 38: Einbettung des GAP-Modells in die TOGAF Architecture Alternatives Method

Ausgangspunkt für die Architekturentwicklung sind die Rahmenbedingungen für die Architektur. Diese sind wie oben beschrieben definiert durch die Vision, Anforderung und die Prinzipien. Und ergeben sich aus den individuellen Anforderungen an die Integration, sowie aus den technischen, betrieblichen und organisatorischen Rahmenbedingungen.

Abgebildet werden die Rahmenbedingungen für die Architektur durch die gewichteten individuellen Anforderungen, sowie die eventuellen Einschränkungen des Optionsraums aufgrund von betrieblichen, technischen oder organisatorischen Gegebenheiten.

Ausgehend von den Anforderungen erfolgt im nächsten Schritt die Analyse und Bewertung der Integrationsarchitekturen. Hierzu erfolgt zunächst eine Kategorisierung und Archetypisierung der Integrationsarchitekturen nach der in Kapitel 3 ausdefinierten Taxonomie entlang der in Abschnitt 2.3 näher beschriebenen Ebenen der IT-Architektur. Diese drei Granularitätsebenen bilden die horizontale Segmentierung der Architektur. Die vertikale Segmentierung erfolgt innerhalb der verschiedenen Granularitätsebenen nach den jeweiligen Merkmalen der Designartefakte.

Grundlage für einen modellgetriebenen Ansatz ist die in Kapitel 3 definierte Taxonomie, da erst hierdurch die strukturierte Vergleichbarkeit von verschiedenen Systemintegrationen ermöglicht wird.

Für die Analyse bestehender Strukturen müssen die zu betrachtenden Integrationsarchitekturen zunächst in ein Modell überführt werden. Davon ausgehend können diese dann in Designartefakte zerlegt und anhand ihrer Merkmale kategorisiert werden. Die Klassifizierung von Integrationsstopologien, -paradigmen, und -methoden ermöglicht dann im nächsten Schritt eine Zuordnung zu einem Archetyp, sowie eine entsprechende Bewertung im Kontext der eingangs definierter Anforderungen.

Für die Auswahl der passenden Integrationsarchitektur für einen bestimmten Anwendungsfall erfolgt eine Bewertung aller Archetypen im Optionsraum. Aus diesem automatisch berechneten Bewertungsergebnis ergibt sich dann ein modellbasierter Vorschlag für passender Integrationsarchitekturen.

Das GAP-Modell ermöglicht im Kontext von TOGAF, dass gute Architekturentscheidungen herbeigeführt werden. So wird die Systemintegration verbessert und die digitale Transformation durch eine bessere Vernetzung interagierender Systeme beschleunigt. Entsprechend der anfangs beschriebenen Motivation wird so innovations- und agilitätsbremsende „Accidental Architecture“ historisch gewachsener Systemlandschaften aufgebrochen. Das GAP-Modell hat somit das Potenzial den Wandel zu optimierten Konnektivitätsökosystemen zu beschleunigen und begünstigt damit die Wertschöpfungssteigerung und erhöhte Agilität durch Digitalisierung mit APIs.

4.5. Zusammenfassung: Das GAP-Modell in Kürze

Das Konzept „GAP-Modell“ beschreibt einen modellgetriebenen Ansatz zum Design von Integrationsarchitekturen. „GAP“ steht als Akronym für „Good Architecture Practice“ und ist angelehnt an den GxP-Begriff aus dem Bereich Qualifizierter Systeme. Es handelt sich dabei um ein automatisierbares Mess- und Entscheidungsmodell für Softwarearchitektur. Hintergrund ist die Modernisierung historisch gewachsener Systemlandschaften im Kontext der digitalen Transformation. Das Institut für deutsche Wirtschaft benennt in seiner aktuellen Studie „Digitalisierungsindex 2022“ fehlende Expertise und begrenzte Ressourcen als Hemmnis für die erfolgreiche Umsetzung der digitalen Transformation. [4, Seite 38] Um dieser begrenzten Kompetenzsouveränität zu begegnen und die digitale Transformation zu beschleunigen entwirft der Autor einen modellbasierten Ansatz für gut designte Integrationsarchitekturen. Hieraus ergeben sich vier Kernanforderungen und das GAP-Modell:

- Automatisierbarkeit für den massenhaften Einsatz
- Einfache Anwendung und Klarheit der Empfehlungen
- Generische Bewertungsmodell zur Bestandsanalyse
- Modellbasierte Unterstützung von Designentscheidungen

Zentraler Aspekt des GAP-Modells ist die Archetypisierung von Architekturen und die Bewertung anhand der Merkmale des jeweiligen Archetyps. Die Zuordnung erfolgt anhand der in Kapitel 3 definierten Taxonomie. Ausgangspunkt für die Definition der Bewertungskriterien ist die ISO 27000, sowie der hierauf basierende BSI-Standard 200 zum IT-Grundschutz. [13, Seite 13]. Auf dieser Grundlage bewertet das GAP-Modell Architekturen in folgenden Dimensionen:

Robustheit: beschreibt, in welchem Maßstab die integrierten Daten und Funktionalitäten verfügbar sind. [13, Seite 12]

Integrität: beschreibt den Grad der Richtigkeit und Vollständigkeit Daten. [13, Seite 5]

Skalierbarkeit: beschreibt die Fähigkeit zur Anpassung von Systemen an einen veränderten Leistungsbedarf.

Kontrolle: beschreibt die Verfügbarkeit von Informationen zum Management von Risiken, sowie die Fähigkeit diese zu beherrschen. [13, Seite 3–4]

Flexibilität: beschreibt die Fähigkeit auf Veränderungen zu reagieren und entsprechende Anpassungen umzusetzen (Resilienz). [76, Seite 14]

Durchsatz: beschreibt das Leistungsvermögen der Kommunikation zwischen integrierten Systemen und ist definiert durch die Menge an Daten, welche in einer bestimmten Zeit übertragen werden, kann. [77]

Latenz: beschreibt die Wartezeit zwischen der Aktualisierung einer Information und der tatsächlichen Datenübertragung. [77]

Die Bewertung jedes Aspektes erfolgt qualitativ auf einer Ordinalskala von *sehr schwach* (1), über *schwach* (2), *stark* (3), bis *sehr stark* (4). Die Abstände zwischen den Bewertungsausprägungen sind dabei gleich groß. Jede Topologie, jedes Paradigma und jede Methode erhält dabei eine eigene Wertung. Die Ermittlung dieser initialen Parameter erfolgt einmalig und Anwendungsfallagnostisch und wird in Kapitel 5 für im Kontext der Fallstudien näher beschrieben.

Im Ersten Schritt der Anwendung des GAP-Modells wird das Anforderungsprofil für den zu bewertenden Anwendungsfall definiert. Hierzu werden die einzelnen Bewertungsdimensionen gewichtet. Aus der Gewichtung ergibt sich dann für jede Dimension eine Minderbewertung, welche von einer passenden Integrationsarchitektur erreicht werden muss.

Das GAP-Modell basiert auf der Archetypisierung von Integrationsarchitekturen als Instrument der merkmalsgetriebenen Einordnung und Vergleichbarkeit von Architekturmustern. Da sich ein Architekturarchetyp aus der Kombination einer bestimmten Integrationstopologie mit einem bestimmten Paradigma und einer bestimmten Methode ergibt, werden die jeweiligen Wertung zusammengenommen und ergeben die Bewertung dieses Archetyps. Für *Robustheit*, *Integrität*, *Kontrolle* und *Flexibilität* ist sowohl der konzeptionelle Aufbau der Integrationen, sowie die technische Implementierung entscheidend, weshalb diese Kriterien für die Bewertung auf allen drei Ebenen (Topologie, Paradigma und Methode) angewendet werden. *Skalierbarkeit* ist vor allem getrieben durch den konzeptionellen Aufbau der Integrationslandschaft und findet daher nur bei der Bewertung der Topologie Anwendung. *Durchsatz* und *Latenz* sind abhängig von der tatsächlichen Implementierung der direkten Systemintegration, weshalb dieser Punkte nur bei der Bewertung von Paradigma und Methode angewendet werden.

Die Gesamtheit aller möglichen Archetypen bildet den Optionsraum für die Auswahl der richtigen Architektur. Dabei können technische, betriebliche und organisatorische Rahmenbedingungen diesen Optionsraum einengen und beispielsweise bestimmte Integrationstopologien ausschließen. Das GAP-Modell vergleicht nun die Bewertungen der einzelnen Archetypen im Optionsraum mit dem Anforderungsprofil ab. Wird in allen Dimensionen der Mindestwert erreicht, so bewertet das GAP-Modell diese Architektur als passend für den Anwendungsfall. Zur weiteren Verfeinerung der Ergebnisse erfolgt zudem eine Bewertung der jeweiligen Architektur, indem die jeweiligen Wertzahlen mit der relativen Gewichtung multipliziert werden.

Für die Bewertung bestehender Implementierungen wird die Integration zunächst einen passenden Archetyp zugeordnet. Die Bewertung des jeweiligen Archetyps bildet dann die Bewertung der Bestandsintegration.

Das GAP-Modell erlaubt somit sowohl die Bewertung bereits implementierter Integrationen als auch die Empfehlung von Entwurfsmustern für neue Anwendungsfälle. Somit kann zum einen Handlungsbedarf bei bestehenden Integrationen schnell und einfach ermittelt werden, zum anderen erleichtert das GAP-Modell die richtige Gestattung der Architektur für neu zu implementierende Integrationen.

Für einen massenhaften Einsatz im Kontext der digitalen Transformation im Enterprise-Umfeld ist das GAP-Modell in das TOGAF Architekturrahmenwerk eingebettet. Abbildung 38 auf Seite 62 zeigt das Konzipierte GAP-Modell und seine Einbettung in das TOGAF Rahmenwerk. Das GAP-Modell ist in dabei eine konkrete Implementierung der TOGAF Architecture Alternatives Method im Kontext der iterativen Architecture Development Method.

5. Parametrisierung des GAP-Modells durch vergleichende Bewertung von Designartefakten

In diesem Kapitel erfolgt die initiale Parametrisierung des GAP-Modells. In diesem Kontext wird die Forschungsfrage FF 3 (*Wie lassen sich Integrationsarchitekturen in Bezug auf die individuellen Anforderungen modellbasiert bewerten?*) weiter beantwortet.

Um das GAP-Modell anwenden zu können, muss dieses zunächst die Wertzahlen für die einzelnen Klassen von Designartefakten ermittelt werden. Hierfür betrachtet der Autor konzeptionelle Vor- und Nachteile aller definierten Klassen von Integrationstopologien, -paradigmen und -methoden. Die Bewertung erfolgt dabei entlang der in Kapitel 4 definierten Bewertungskriterien und liefert die initialen Parameter für die Anwendung des GAP-Modells.

5.1. Enterprise-Ebene: Gegenüberstellung ausgewählter Integrationstopologien

Integrationsarchitekturen werden auf Enterprise-Ebene verschiedenen Topologien zugeordnet.

Tabelle 10 zeigt zur Orientierung die betrachteten Integrationsparadigmen, welche sich aus Klassifizierung entsprechend der in Kapitel 3 beschriebenen Taxonomie ergeben. Die Integrationstopologie beschreibt dabei die konzeptionelle Schematik, nach der Integrierte Systeme miteinander verbunden sind. Die Klassifizierung erfolgt anhand der zentralen, oder verteilten Kontrollstrategie, sowie der exklusiven, bzw. distributiven Kommunikationsflexibilität.

Topologie (Enterprise-Ebene)		Kommunikationsflexibilität	
		<i>exklusiv</i>	<i>distributiv</i>
Kontrollstrategie	<i>zentral</i>	Hub	Bus
	<i>verteilt</i>	Tree	Mesh

Tabelle 10: Integrationstopologien im GAP-Modell (Enterprise-Ebene)

Ausgehend von den oben beschriebenen Merkmalen analysiert der Autor Vor- und Nachteile der jeweiligen Topologien. Diese ergeben sich vor allem dadurch, wie sich die Pfade zum Informationsaustausch zwischen den Systemen unterscheiden und an welcher Stelle Transformations- und Integrationslogiken abgebildet werden. Die vergleichende Bewertung auf Enterprise-Ebene erfolgt entlang der in Abschnitt 4.2 definierten Dimensionen des Bewertungsmodells.

5.1.1. Integration-Hub

Merkmale: zentrale Kontrollstrategie, exklusive Kommunikation

Robustheit: sehr schwach

Ein Integration-Hub fußt auf der Implementierung einer zentralen Komponente. Da die gesamte Kommunikation und Datentransformation über den Hub realisiert wird, kann die gesamte Integrationslandschaft bei konzeptionellem oder technischem Problem kollabieren (Single-Point-of-Failure). Der Autor bewertet daher die Robustheit bei Hub-Topologien mit der Wertzahl 1 (*sehr schwach*).

Integrität: sehr stark

Aufgrund der zentralisierten Datenhaltung im Information-Hub existiert ein klarer Referenzpunkt für alle verbundenen Systeme. Bei Konflikten können Informationen somit immer mit dem Hub synchronisiert werden. Zudem treten alle Akteure lediglich mit dem Hub in Interaktion, welcher Informationen und Interaktionen als konsolidierte Services anbieten kann. Die Komplexität der einzelnen Integrationen wird für die verbundenen Systeme ausgeblendet und es kann keine Abweichungen bei Transformations- und Integrationslogiken geben. Der Autor bewertet daher die Integrität bei Hub-Topologien mit der Wertzahl 4 (*sehr stark*).

Skalierbarkeit: schwach

Die Skalierung des Konnektivitätsökosystems ist durch die Möglichkeiten des Information-Hubs begrenzt. Zusätzlich zu den Beschränkungen der integrierten Systeme muss jegliche Kommunikation über eine zentrale Komponente abgewickelt werden, welche sich sowohl technisch als auch konzeptionell zum Flaschenhals entwickeln kann. Für jede Integration muss eine vollumfängliche Einbindung in den Integration-Hub erfolgen, welche zentral umgesetzt werden muss. Ist der Zugriff auf diese zentralen Fähigkeiten begrenzt, so schränkt dies die Skalierbarkeit stark ein. Der Autor bewertet daher die Skalierbarkeit bei Hub-Topologien mit der Wertzahl 2 (*schwach*).

Kontrolle: sehr stark

Über den Integration-Hub können alle Informationsflüsse erfasst, gesteuert und kontrolliert werden. Durch diese zentrale Kontrollstrategie kann eine sehr starke Kontrolle über das Konnektivitätsökosystem erreicht werden. Der Autor bewertet daher die Kontrolle bei Hub-Topologien mit der Wertzahl 4 (*sehr stark*).

Flexibilität: stark

Bei Bedarf können einzelne Komponenten einfach aus dem Ökosystem entfernt, oder ausgetauscht werden, ohne dass dies direkte Auswirkungen auf die anderen verbundenen Systeme hätte. Der Autor bewertet daher die Flexibilität bei Hub-Topologien mit der Wertzahl 3 (*stark*).

5.1.2. Integration-Bus

Merkmale: zentrale Kontrollstrategie, distributive Kommunikation

Robustheit: schwach

Beim Integration-Bus tauschen die jeweiligen Systeme die Informationen zwar direkt miteinander aus, nutzen aber eine zentrale Middleware zur Abwicklung der Kommunikation. Somit ist der Bus zwar robust gegenüber konzeptionellen Problemen in einzelnen Integrationen, die Robustheit ist jedoch durch die erforderliche Middleware begrenzt. Der Autor bewertet daher die Robustheit bei Bus-Topologien mit der Wertzahl 2 (*schwach*).

Integrität: schwach

Die distributive Kommunikation beim Integration-Bus führt zu einer Vielzahl von konzeptionellen Direktverbindungen zwischen den Systemen, ohne dass es einen eindeutigen Bezugspunkt für Transformationen und Logiken gibt. Hierdurch entsteht ein sehr komplexes Ökosystem in der es leicht zu Integrationskonflikten kommen kann. Der Autor bewertet daher die Integrität bei Bus-Topologien mit der Wertzahl 2 (*schwach*).

Skalierbarkeit: stark

Da die jeweiligen Integrationen gekapselt voneinander implementiert und betrieben werden, kann das Konnektivitätsökosystem beliebig um weitere Integrationen erweitert werden. Lediglich die Kapazität des Integration-Bus als Middleware zur Abwicklung der Kommunikation zwischen den Systemen begrenzt die Skalierbarkeit der Integrationslandschaft. Der Autor bewertet daher die Skalierbarkeit bei Bus-Topologien mit der Wertzahl 3 (*stark*).

Kontrolle: stark

Aufgrund der zentralen Kontrollstrategie beim Integration-Bus kann die gesamte Kommunikation an zentraler Stelle beherrscht werden. Da alle Informationen über den Bus ausgetauscht werden, besteht eine sehr gute Sichtbarkeit über die Informationsflüsse im Kommunikationsökosystem. Der Autor bewertet daher die Kontrolle bei Bus-Topologien mit der Wertzahl 3 (*stark*).

Flexibilität: sehr stark

Systeme könne beliebig zum Integration-Bus hinzugefügt, oder entfernt werden. Da alle Informationsflüsse über den Bus bekannt sind, können Änderungen an bestehenden Integrationen gut geplant werden. Die distributive Kommunikation ermöglicht eine, Änderungen isoliert umzusetzen, ohne dass sich diese auf andere Integrationen auswirken. Der Autor bewertet daher die Flexibilität bei Bus-Topologien mit der Wertzahl 4 (*sehr stark*).

5.1.3. Integration-Tree

Merkmale: verteilte Kontrollstrategie, exklusive Kommunikation

Robustheit: stark

Beim Integration-Tree gibt es keine zentrale Komponente, so dass der Ausfall eines einzelnen Systems nicht zwangsläufig zu einem Ausfall des gesamten Trees führt. Dabei ist jedoch zu beachten, dass alle Knoten entlang eines Pfades nach dem ausgefallenen System ebenfalls ausfallen. Der Autor bewertet daher die Robustheit bei Tree-Topologien mit der Wertzahl 3 (*stark*).

Integrität: stark

Die Integrität im Integration-Tree wird nicht durch eine zentrale Instanz erreicht. Jedoch beugt die Topologie Synchronisierungskonflikten und Asynchronitäten vor. Aufgrund der exklusiven Kommunikation werden Transformations- und Integrationslogik für jedes System nur genau einmal implementiert. Auch werden Informationen nur über genau einen Pfad ausgetauscht und nicht aus verschiedenen Systemen zusammengeführt, so dass es hierbei nicht zu Problemen kommen kann. Nichtsdestotrotz kann ein Fehler entlang des Integrationspfades zu Integritätsproblemen in den abhängigen Systemen führen. Der Autor bewertet daher die Integrität bei Tree-Topologien mit der Wertzahl 3 (*stark*).

Skalierbarkeit: sehr stark

Der Integration-Tree skaliert sehr gut, da der Baum aufgrund fehlender Beschränkungen nahezu beliebig wachsen kann. Das Merkmal der exklusiven Kommunikation führt dabei zu einer klaren Struktur des Konnektivitätsökosystems und begrenzt die Komplexität, welche beherrscht werden muss. Der Autor bewertet daher die Skalierbarkeit bei Tree-Topologien mit der Wertzahl 4 (*sehr stark*).

Kontrolle: stark

Obwohl im Integration-Mesh eine zentrale Kontrollkomponente fehlt, sind durch die klare Struktur der Integrationslandschaft Abhängigkeiten einfach zu erkennen. Der Autor bewertet daher die Kontrolle bei Tree-Topologien mit der Wertzahl 3 (*stark*).

Flexibilität: sehr schwach

Eine Systemänderung im Integration-Tree beeinflusst alle Systeme entlang des Informationspfades. Da es keine Zusammenführung von Informationen, gibt und der Informationsfluss nur in eine Richtung vorgesehen ist, muss bei diesbezüglichen Änderungen die Baumstruktur umgebaut werden. Es empfiehlt sich daher Schnittstellen möglichst Systemagnostisch zu definieren, so dass die Applikationen dahinter flexibel ausgetauscht werden können, ohne dass die jeweiligen Integrationen umfassend überarbeitet werden müssen. Der Autor bewertet daher die Flexibilität bei Tree-Topologien mit der Wertzahl 1 (*sehr schwach*).

5.1.4. Integration-Mesh

Merkmale: verteilte Kontrollstrategie, distributive Kommunikation

Robustheit: sehr stark

Im Integration-Mesh sind alle Integrationen individuell implementiert, so dass es keine zentrale kritische Komponente gibt. Ausfälle sind somit nur die Integrationen entlang eines betroffenen Pfades beschränkt, wobei das Netzwerk an sich intakt bleibt. Der Autor bewertet daher die Robustheit bei Mesh-Topologien mit der Wertzahl 4 (*sehr stark*).

Integrität: sehr schwach

Die Vielzahl an Direktverbindungen wirken sich bei wachsender Komplexität der Integrationslandschaft potenziell negativ auf die Integrität aus. Da Integrationen mit unterschiedlicher Logik implementiert werden können und es keine zentrale Referenz gibt, kann es schnell zu Asynchronitäten und Konflikten kommen. Der Autor bewertet daher die Integrität bei Mesh-Topologien mit der Wertzahl 1 (*sehr schwach*).

Skalierbarkeit: stark

Das Integration-Mesh skaliert gut, da es außer den lokalen Einschränkungen der jeweils angebundenen Systeme keine Einschränkungen gibt. Das Netzwerk kann einfach ohne Auswirkungen auf die bereits bestehenden Integrationen durch die Einführung neuer Knoten und Kanten erweitert werden. Hierbei ist jedoch hervorzuheben, dass mit steigender Anzahl und Heterogenität der Integrationen auch die Komplexität des Konnektivitätsökosystems steigt. Der Autor bewertet daher die Skalierbarkeit bei Mesh-Topologien mit der Wertzahl 3 (*stark*).

Kontrolle: sehr schwach

Da im Integration-Mesh die zentrale Komponente fehlt, ist die Kontrolle über die jeweiligen Integrationen eine Herausforderung. Mit wachsender Zahl der integrierten Systeme wächst auch die Komplexität der Systemlandschaft, da es viele Direktverbindungen und Abhängigkeiten gibt. Zudem sind Informationsflüsse ohne zentralen Kontrollmechanismus nur sehr schwer nachzuvollziehen. Der Autor bewertet daher die Kontrolle bei Mesh-Topologien mit der Wertzahl 1 (*sehr schwach*).

Flexibilität: sehr stark

Das Integration-Mesh lässt sich beliebig anpassen, da jede einzelne Integration entsprechend der distributiven Kommunikation unabhängig implementiert ist. Zudem können Systeme an beliebiger Stelle in das Netzwerk aufgenommen werden. Auch das Anpassen der Informationsflüsse selbst ist einfach möglich. Dabei ist zu beachten, dass jede Systemänderung Auswirkungen auf alle direkt oder indirekt verbundenen Systeme hat. Jedoch bleibt dieser Effekt aufgrund der distributiven Kommunikationsstrategie auf den jeweiligen Informationspfad begrenzt. Der Autor bewertet daher die Flexibilität bei Mesh-Topologien mit der Wertzahl 4 (*sehr stark*).

5.2. Applikationsebene: Gegenüberstellung ausgewählter Integrationsparadigmen

Integrationsarchitekturen werden auf Applikationsebene verschiedenen Paradigmen zugeordnet.

Paradigma (Applikationsebene)		Ausführung der Informationsintegration	
		<i>materialisiert</i>	<i>virtuell</i>
Initiierung des Informationsaustausches	<i>proaktiv (pull)</i>	Storage-Driven	Service-Driven
	<i>reaktiv (push)</i>	Event-Driven	

Tabelle 11 zeigt zur Orientierung die betrachteten Integrationsparadigmen, welche sich aus Klassifizierung entsprechend der in Kapitel 3 beschriebenen Taxonomie ergeben. Das Integrationsparadigma beschreibt dabei, wie die Kommunikation zwischen zwei integrierten Systemen gestaltet wird. Die Klassifizierung erfolgt anhand der proaktiven, bzw. reaktiven Initiierung des Informationsaustausches, sowie der materialisierten, oder virtualisierten Ausführung der Informationsintegration.

Paradigma (Applikationsebene)		Ausführung der Informationsintegration	
		<i>materialisiert</i>	<i>virtuell</i>
Initiierung des Informationsaustausches	<i>proaktiv (pull)</i>	Storage-Driven	Service-Driven
	<i>reaktiv (push)</i>	Event-Driven	

Tabelle 11: Integrationsparadigmen im GAP-Modell (Applikationsebene)

Der Autor geht in Folgendem auf Stärken und Schwächen ein, welche aus den jeweiligen konzeptionellen Rahmenbedingungen der oben beschriebenen Merkmale folgen. Es wird insbesondere die Art und Weise betrachtet, wie Consumer die Daten nutzen und welche Akteure die Kommunikation zwischen dem System initiieren. Die vergleichende Bewertung auf Applikationsebene erfolgt entlang der in Abschnitt 4.2 definierten Dimensionen des Bewertungsmodells.

5.2.1. Storage-Driven-Integration

Merkmale: Proaktive Initialisierung der Kommunikation, materialisierte Ausführung der Informationsintegration

Robustheit: sehr stark

Storage-Driven-Integrations sind sehr robust, da durch die materialisierte Informationsintegration jederzeit eine physische Kopie der Daten im Zielsystem vorgehalten wird. Die Abhängigkeit vom Provider ist dadurch auf die Extraktion begrenzt. Auch Consumer müssen nur für den Load verfügbar sein, wodurch eine zeitliche Entkopplung möglich ist. Der Autor bewertet daher die Robustheit bei Storage-Driven-Integrations mit der Wertzahl 4 (*sehr stark*).

Integrität: schwach

Da Daten physisch kopiert werden und keine Virtualisierung stattfindet, entstehen zwangsläufig Dubletten. Ohne zusätzliche Maßnahmen in Bezug auf Data-Ownership kann dies zu Asynchronitäten und der Korruption von Daten führen. Somit ist die Integrität bei speicherbasierten Integrationen problematisch. Der Autor bewertet daher die Integrität bei Storage-Driven-Integrations mit der Wertzahl 2 (*schwach*).

Kontrolle: stark

Aufgrund der proaktiven Initialisierung der Kommunikation kann der Provider den Zugriff auf die von ihm übermittelten Informationen steuern. So kann über granulare Zugriffsrechte in beschränktem Maße Kontrolle im Konnektivitätsökosystem ausgeübt werden. Der Autor bewertet daher die Kontrolle bei Storage-Driven-Integrations mit der Wertzahl 3 (*stark*).

Flexibilität: sehr schwach

Ohne Abstraktion müssen Änderungen an den Datenstrukturen bei einer Änderung entlang des Integrationspfades in allen integrierten Systemen angepasst werden. Der Autor bewertet daher die Flexibilität bei Storage-Driven-Integrations mit der Wertzahl 1 (*sehr schwach*).

Durchsatz: stark

Abhängig von der Art der Implementierung kann bei speicherbasierten Integrationen ein hoher Datendurchsatz erreicht werden, so dass auch die komplette Datenbasis innerhalb einer Iteration kopiert werden kann. Der Autor bewertet den Durchsatz von Storage-Driven-Integrations mit der Wertzahl 3 (*stark*).

Latenz: sehr schwach

Der große Nachteil speicherbasierten Integrationen ist, dass Änderungen im Quellsystem nur zeitversetzt in die Zielsysteme übertragen werden. Dies ist von Nachteil, da die integrierten Daten zum Zeitpunkt der Verarbeitung durch den Consumer nicht mehr aktuell sind. Der Autor bewertet daher die Latenz bei Storage-Driven-Integrations mit der Wertzahl 1 (*sehr schwach*).

5.2.2. Service-Driven-Integration

Merkmale: Proaktive Initialisierung des Informationsaustausches, virtualisierte Ausführung der Informationsintegration

Robustheit: sehr schwach

Durch die fehlende Persistierung von Daten fehlt die Rückfallebene im Falle eines Providerausfalls. Somit ist die virtualisierte Ausführung der Informationsintegration in Bezug auf Robustheit ein Nachteil. Die schwächste Komponente entlang des Integrationspfades ist dabei ausschlaggebend. Der Autor bewertet daher die Robustheit bei Service-Driven mit der Wertzahl 1 (*sehr schwach*)

Integrität: sehr stark

Durch die Virtualisierung werden Dubletten vermieden. Aufgrund der bedarfsgesteuerten Initialisierung der Integration verfügt das Consumer-System zudem immer über die aktuellen Informationen, so dass Asynchronität vermieden wird. Mit dem exklusiven Verbleib der Daten in den Quellsystemen wird außerdem eine klare Data-Ownership erreicht. Dies führt zu einer hohen Integrität. Der Autor bewertet daher die Integrität bei Service-Driven-Integrations mit der Wertzahl 4 (*sehr stark*).

Kontrolle: stark

Provider können die Beantwortung der proaktiv initiierten Kommunikation präzise steuern und kontrollieren. Der Autor bewertet daher die Kontrolle bei Service-Driven-Integrations mit der Wertzahl 3 (*stark*).

Flexibilität: sehr stark

Die konsequente Entkopplung von Schnittstelle und Implementierung ermöglicht eine sehr flexible Anpassung der einzelnen Komponenten, ohne dass diese einen negativen Einfluss auf die verbundenen Systeme hat. Auch reduziert die Kapselung von Daten und Funktionalität in Services die Komplexität der einzelnen Integration. Zudem können bestehende Funktionalitäten durch Serviceorchestrierung sehr effizient zu neuen Produkten zusammengesetzt werden. Der Autor bewertet daher die Flexibilität bei Service-Driven-Integrations mit der Wertzahl 4 (*sehr stark*).

Durchsatz: sehr schwach

Obwohl Daten in hoher Geschwindigkeit ausgetauscht werden können, müssen diese aufgrund der Virtualisierung der Informationsintegration bei Bedarf immer wieder neu abgefragt werden. Der Autor bewertet daher den Durchsatz bei Service-Driven-Integrations mit der Wertzahl 1 (*sehr schwach*).

Latenz: sehr stark

Die proaktive Initiierung der Kommunikation ermöglicht eine bedarfsorientierte Steuerung der Informationsintegration, bei der die benötigten Informationen auf Anfrage sofort vom Provider zur Verfügung gestellt werden. Der Autor bewertet daher die Latenz bei Service-Driven-Integrations mit der Wertzahl 4 (*sehr stark*).

5.2.3. Event-Driven-Integration

Merkmale: Reaktive Initialisierung des Informationsaustausches

Robustheit: schwach

Bei Event-Driven-Integrations initiiert der Provider die Kommunikation und informiert proaktiv mit einem Event über erfolgte Änderungen. Eine korrekte Verarbeitung dieser Informationen setzt voraus, dass alle Consumer-Systeme diese Nachricht erhalten. Durch den Einsatz von Queues können Systeme zwar nach einem temporären Ausfall wieder in den Integrationsfluss einsteigen, dabei müssen jedoch Änderungen während der Auszeit aufgearbeitet werden. Der Autor bewertet daher die Robustheit bei Event-Driven-Integrations mit der Wertzahl 2 (*schwach*).

Integrität: stark

Alle Informationen werden als Messages über entsprechende Topics und Queues in Echtzeit an alle verbundenen Systeme verteilt. Somit verfügen alle Akteure im Konnektivitätsökosystem zu jeder Zeit über den aktuellen Informationsstand. Der Autor bewertet daher die Integrität bei Event-Driven-Integrations mit der Wertzahl 3 (*stark*).

Kontrolle: sehr schwach

Anstatt einer Vereinfachung durch Kapselung der Informationen, werden diese in eine Vielzahl von Events fragmentiert. Dies erhöht zwar die Flexibilität der Integrationen, erschwert allerdings die Kontrolle über einzelne Verbindungen enorm. Der Autor bewertet daher die Kontrolle bei Event-Driven-Integrations mit der Wertzahl 1 (*sehr schwach*).

Flexibilität: sehr stark

Durch die Zuordnung von Events zu Topics sind alle Aktivitäten logisch abstrahiert und system-agnostischen abgebildet. Die Anzahl der verbundenen Systeme ist dabei unerheblich. Der Autor bewertet daher die Flexibilität bei Event-Driven-Integrations mit der Wertzahl 4 (*sehr stark*).

Durchsatz: stark

Da Systeme sofort auf jegliche Änderungen reagieren und die Daten entsprechend weitergeben, entsteht ein kontinuierlicher Datenfluss, bei welchem selbst mit sehr kleinen Datenmengen pro Nachricht in einem bestimmten Zeitfenster sehr viele Daten übertragen werden können. Dabei werden jedoch stets nur die Änderungen übertragen. Gleichzeitig ist die Kommunikation sehr effizient, da die Komponenten nur bei Änderungen aktiv werden. Der Autor bewertet daher den Durchsatz bei Event-Driven-Integrations mit der Wertzahl 3 (*stark*).

Latenz: sehr stark

Mit Event-Driven-Integration werden Informationen direkt beim Eintreten eines Events sofort in Echtzeit übermittelt. Die Verarbeitung dieser Informationen erfolgt dann durch die Consumer in den individuell avisierten Zeiträumen. Der Autor bewertet daher die Latenz bei Event-Driven-Integrations mit der Wertzahl 4 (*sehr stark*).

5.3. Technische Ebene: Gegenüberstellung ausgewählter Integrationsmethoden

Integrationsmethoden beschreiben die technische Implementierung der System-zu-System-Kommunikation.

Methode (Technische Ebene)		Format der Datenendeinrichtung		
		<i>dateibasiert</i>	<i>webbasiert</i>	<i>tabellenbasiert</i>
Kommunikationsart der Übertragungs- einrichtung	<i>push</i>	File-Exports	Webhooks	Change-Data-Cap- ture
	<i>pull</i>	File-Imports	Webservices	Database-Connec- tion
Systemnativität der Übertragungs- einrichtung	<i>intern</i>	Native API		
	<i>extern</i>	API-Wrapper/-Mediator		

Tabelle 12 zeigt zur Orientierung die betrachteten Integrationsmethoden, welche sich aus Klassifizierung entsprechend der in Kapitel 3 beschriebenen Taxonomie ergeben. Auf technischer Ebene erfolgt dabei eine Klassifizierung von Integrationsarchitekturen nach dem Format der Datenendeinrichtung (dateibasiert, webbasiert, oder tabellenbasiert), sowie nach der Kommunikationsart der Übertragungseinrichtung (push, oder pull). Zusätzlich wird entsprechend der Systemnativität der Übertragungseinrichtung zwischen Native API und API-Wrapper/-Mediator unterschieden.

Methode (Technische Ebene)		Format der Datenendeinrichtung		
		<i>dateibasiert</i>	<i>webbasiert</i>	<i>tabellenbasiert</i>
Kommunikationsart der Übertragungs- einrichtung	<i>push</i>	File-Exports	Webhooks	Change-Data-Cap- ture
	<i>pull</i>	File-Imports	Webservices	Database-Connec- tion
Systemnativität der Übertragungs- einrichtung	<i>intern</i>	Native API		
	<i>extern</i>	API-Wrapper/-Mediator		

Tabelle 12: Integrationsmethoden im GAP-Modell (Technische Ebene)

In folgendem betrachtet der Autor welche Vor- und Nachteile die jeweiligen technischen Integrationsansätze auszeichnet, sowie welchen Einfluss es hat, ob die entsprechenden technischen Fähigkeiten Teil des integrierten Systems sind, oder ob diese nachträglich hinzugefügt sind. Die vergleichende Bewertung auf technischer Ebene erfolgt entlang der in Abschnitt 4.2 definierten Dimensionen des Bewertungsmodells.

5.3.1. File-Exports

Merkmale: dateibasierte push-Integration

Robustheit: sehr stark

Bei File-Exports kann die Gesamtheit aller Daten innerhalb einer Datenübermittlung in einer Datei aus dem System gesendet werden. Die komplette Aktivität wird damit auf der Provider-Seite durchgeführt und vom Consumer entkoppelt, so dass die Extraktion nicht durch Verbindungsabbrüche, oder Ähnliches beeinträchtigt werden kann. Der Autor bewertet daher die Robustheit bei File-Exports mit der Wertzahl 4 (*sehr stark*).

Integrität: stark

Bei File-Exports können benötigten Daten in einer Datei zusammengefasst werden, so dass diese immer in sich synchron sind. Gleichzeitig stellt ein File-Export jedoch auch immer nur einen Snapshot zum Zeitpunkt des Exports dar, so dass die Synchronität zwischen verschiedenen Akteuren nichts zwangsläufig gegeben ist. Der Autor bewertet daher die Integrität bei File-Exports mit der Wertzahl 3 (*stark*).

Kontrolle: schwach

Ohne zusätzliche Maßnahmen ist bei File-Exports nicht nachvollziehbar, wer Zugriff auf die geteilten Informationen erlangt. Nach der Extraktion kann die zur Verfügung gestellte Datei somit unkontrolliert genutzt werden. Der Autor bewertet daher die Kontrolle bei File-Exports mit der Wertzahl 2 (*schwach*).

Flexibilität: stark

Wenn File-Exports keinen strengen Formatanforderungen folgen, können Anforderungsänderungen mit geringem Umfang umgesetzt werden. Der Autor bewertet daher die Flexibilität bei File-Exports mit der Wertzahl 3 (*stark*).

Durchsatz: sehr stark

Da File-Exports alle benötigten Daten in einer Datei bereitstellen können, können diese sehr effizient übertragen werden. So ist zum Beispiel der Export eines Snapshots einer ganzen Datenbank in einem sehr kurzen Zeitfenster möglich. Der Autor bewertet daher den Durchsatz bei File-Exports mit der Wertzahl 4 (*sehr stark*).

Latenz: schwach

Da beim Erstellen der Datei für den File-Export der jeweilige Ist-Zustand unveränderlich abgebildet wird, ist bei dieser Integrationsmethode per-se mit einer zeitlichen Lücke zwischen Bereitstellung und Verarbeitung der Daten zu rechnen, womit die Latenz als schwach zu bewerten ist. Die Integration von Funktionalität ist somit auch nur sehr eingeschränkt möglich. Der Autor bewertet daher die Latenz bei File-Exports mit der Wertzahl 2 (*schwach*).

5.3.2. File-Imports

Merkmale: dateibasierte pull-Integration

Robustheit: sehr stark

Analog zu File-Exports sind auch File-Imports sehr robust, da die Daten in Ihrer Gesamtheit in einer Datei gesammelt übertragen werden können. Consumer verfügen somit sofort über alle benötigten Daten, so dass ein Ausfall des Providers zum Zeitpunkt des Imports nicht relevant ist. Der Autor bewertet daher die Robustheit bei File-Imports mit der Wertzahl 4 (*sehr stark*).

Integrität: sehr schwach

Aufgrund der Art der Kommunikationsart kann der Consumer nicht sicherstellen, dass er den aktuellen Datenstand erhält, so dass Datenkollisionen und Asynchronitäten wirksam abgefangen werden müssen. So muss mithilfe einer entsprechenden Import-Logik gesichert werden, dass die Daten in Ihrer Gesamtheit nicht kompromittiert werden. Zusätzlich können je nach Dateiformat kleinste Formatierungsfehler zu schweren Fehlern führen. Der Autor bewertet daher die Integrität bei File-Imports mit der Wertzahl 1 (*sehr schwach*).

Kontrolle: schwach

Wie oben beschrieben ermöglichen File-basierte Integrationsmethoden nur eine begrenzte Kontrolle, da genauer gesteuert werden müsste, wer Zugriff auf welche Dateien erhält. Der Autor bewertet daher die Kontrolle bei File-Imports mit der Wertzahl 2 (*schwach*).

Flexibilität: stark

File-Imports sind durchaus flexibel, da keine besonderen Anforderungen an das bereitzustellende Format gestellt werden. Die Anpassungen sind jedoch mit einigem Aufwand für die Anpassung der Import- und Exportlogiken inklusive Parser auf Provider- und Consumer-Seite verbunden. Der Autor bewertet daher die Flexibilität bei File-Imports mit der Wertzahl 3 (*stark*).

Durchsatz: sehr stark

File-Imports ermöglichen einen sehr starken Durchsatz, da alle relevanten Informationen in einer Datei übertragen werden können. Zudem ermöglicht die Kommunikationsart, dass die Datei bereits vorbereitet wird und zum benötigten Zeitpunkt sofort verfügbar ist. Die Durchführungszeit begrenzt sich damit auf die Zeit, welche für die reine Übertragung der Datei notwendig ist. Der Autor bewertet daher den Durchsatz bei File-Imports mit der Wertzahl 4 (*sehr stark*).

Latenz: sehr schwach

Bei File-Imports liegt zwischen der Erfassung und Bereitstellung der Daten durch den Provider, sowie dem Import durch den Consumer eine beliebig lange Zeit. Somit sind File-Imports zwar für den Austausch statischer Daten geeignet, eignen sich jedoch nicht für dynamische Integrationen, welche eine schnelle Kommunikation zwischen den Akteuren voraussetzt. Der Autor bewertet daher die Latenz bei File-Imports mit der Wertzahl 1 (*sehr schwach*).

5.3.3. Webhooks

Merkmale: webbasierte push-Integration

Robustheit: sehr schwach

Bei der webbasierte push-Integration über Webhooks eröffnet der Provider einen Kommunikationspfad mit dem Consumer. Diese Verbindung muss für die Zeit der Übertragung aktiv bleiben. Bei einem Ausfall einer Komponente bricht die Übertragung. Der Autor bewertet daher die Robustheit bei Webhooks mit der Wertzahl 1 (*sehr schwach*).

Integrität: sehr stark

Webhooks ermöglichen einen sofortigen zielgenauen Austausch von Daten. Da Provider unverzüglich Änderungen kommunizieren können, lässt sich die Datenbasis zuverlässig synchronisieren, womit Konflikte vermieden werden. Der Autor bewertet daher die Integrität bei Webhooks mit der Wertzahl 4 (*sehr stark*).

Kontrolle: sehr stark

Mit Webhooks kann der Provider Datenflüsse sehr genau steuern und bestimmt, welche Daten zu welcher Zeit an welche Endpunkte kommuniziert werden. Der Autor bewertet daher die Kontrolle bei Webhooks mit der Wertzahl 4 (*sehr stark*).

Flexibilität: sehr stark

Bei einer Integration mit Webhooks können Datenflüsse auf Provider-Seite jederzeit angepasst werden. Dieser steuert, welche Daten mit welcher Struktur an welchen Endpunkt kommuniziert werden. Protokolle und Datenstrukturen ermöglichen dabei eine Vielzahl von ein- und mehrdimensionalen Datentypen. Der Autor bewertet daher die Flexibilität bei Webhooks mit der Wertzahl 4 (*sehr stark*).

Durchsatz: schwach

Webbasierten Kommunikation nutzt das HTTP, so dass die mögliche Größe der Datenpakete pro Interaktion begrenzt sind. Webhooks eignen sich gut für die schnelle zielgerichtete Übertragung kleiner Daten. Die Übertragung großer Datenmengen hingegen erfordert eine Unterteilung in eine Vielzahl von Paketen. Der Autor bewertet daher den Durchsatz bei Webhooks mit der Wertzahl 2 (*schwach*).

Latenz: sehr stark

Webhooks ermöglichen die sofortige Kommunikation von Datenänderungen. Die verarbeiteten Informationen sind somit sehr aktuell, da die zeitliche Lücke zwischen Informationen in Quell- und Zielsystem sehr klein ist. Der Autor bewertet daher die Latenz bei Webhooks mit der Wertzahl 4 (*sehr stark*).

5.3.4. Webservices

Merkmale: webbasierte pull-Integration

Robustheit: schwach

Bei Webservices eröffnet der Consumer einen Kommunikationspfad mit dem Consumer. Eine Integration ist somit nur möglich, wenn alle Komponenten zu jeder Zeit verfügbar sind. Der Ausfall eines Consumers wirkt sich nicht zwangsläufig negativ auf die Funktionalität des Providers aus. Consumer jedoch sind vom Serviceangebot des Providers abhängig. Auch wenn Integrationen mit Webservices so implementiert werden sollten, dass der Ausfall eines einzelnen Services nicht den Ausfall des Systems bedeutet, so besteht doch zu jeder Zeit eine generelle Abhängigkeit von Provider. Der Autor bewertet daher die Robustheit bei Webservices mit der Wertzahl 2 (*schwach*).

Integrität: sehr stark

Über Webservices können Daten und Funktionalität punktgenau und unmittelbar integriert werden. Daten müssen nicht dupliziert werden und können stets direkt aus den Quellsystemen angefragt werden. Durch diese Virtualisierung sind Asynchronitäten ausgeschlossen. Der Autor bewertet daher die Integrität bei Webservices mit der Wertzahl 4 (*sehr stark*).

Kontrolle: sehr stark

Webservice-Provider bestimmen jederzeit, welche Daten und Funktionalitäten über welche Endpunkte an welche Consumer zur Verfügung gestellt werden. Die Übertragung und Weitergabe von Daten ist somit sehr gut kontrollierbar, so dass die Komplexität der Systemlandschaft, sowie der einzelnen Integrationen gut beherrschbar ist. Der Autor bewertet daher die Kontrolle bei Webservices mit der Wertzahl 4 (*sehr stark*).

Flexibilität: sehr stark

Über die Service-Endpunkte ergibt sich automatisch eine Abstraktionsschicht, welche eine spätere Anpassung zu jeder Zeit deutlich vereinfacht. Zudem ist es für den Provider irrelevant, welche und wie viele Consumer die zur Verfügung gestellten Services nutzen, so dass eine Anpassung innerhalb des Kommunikationsökosystems sehr einfach möglich ist. Der Autor bewertet daher die Flexibilität bei Webservices mit der Wertzahl 4 (*sehr stark*).

Durchsatz: schwach

Genau wie Webhooks, nutzen Webservices zur Kommunikation http. Auch wenn es möglich ist, größere Payloads über Webservices auszutauschen, ist der Datendurchsatz innerhalb einer Interaktion begrenzt. Der Autor bewertet daher den Durchsatz bei Webservices mit der Wertzahl 2 (*schwach*).

Latenz: sehr stark

Bei einer Integration mit Webservices lassen sich Daten und Funktionalität beinahe in Echtzeit einbinden und Informationen ohne Bruch integrieren. Der Autor bewertet daher die Latenz bei Webservices mit der Wertzahl 4 (*sehr stark*).

5.3.5. Change-Data-Capture

Merkmale: tabellenbasierte push-Integration

Robustheit: stark

Bei der proaktiven Übertragung von Informationen auf Datenbankebene verfügt das Zielsystem im Falle eines Ausfalls immer über den aktuellen Datensatz. Einschränkend wirkt, dass für den Einsatz direkter Integration über Datenbankverbindungen Provider und Consumer zum Übertragen der Änderungen aktiv zur Verfügung stehen müssen. Der Autor bewertet daher die Robustheit bei Change-Data-Capture mit der Wertzahl 3 (*stark*).

Integrität: stark

Durch den die sofortige Kommunikation von Änderungen werden Asynchronitäten verhindert. Einschränkend wirkt, dass durch das Verpassen von Änderungsbenachrichtigungen ein Problempotenzial für Datenkonflikte besteht. Der Autor bewertet daher die Integrität bei Change-Data-Capture mit der Wertzahl 3 (*stark*).

Kontrolle: stark

Integrationen mit Change-Data-Capture ermöglichen eine starke Kontrolle über die Systemintegration, da Zeitpunkt und Ziel der Integrationen jederzeit beim Provider liegen. Zwar können geteilte Daten aus den Zielsystemen wiederum weiterverteilt werden, jedoch können die Datenflüsse vom Provider ausgehend zielgenau gesteuert werden. Der Autor bewertet daher die Kontrolle bei Change-Data-Capture mit der Wertzahl 3 (*stark*).

Flexibilität: schwach

Integrationen auf Datenbankebene realisiert sind verfügen über keine Abstraktionsebene. Daher müssen Änderungen an Datenstrukturen oder Technologie mit der Implementierung der Integration aufwändig abgefangen werden. Der Autor bewertet daher die Flexibilität bei Change-Data-Capture mit der Wertzahl 2 (*schwach*).

Durchsatz: stark

Da die Daten ohne Umwege auf Datenbankebene ausgetauscht werden und die entsprechenden Protokolle und Konnektoren für den Austausch großer Datenmengen konzipiert sind, ist ein hoher Durchsatz möglich. Einschränkend wirkt, dass aufgrund der fehlenden Abstraktion Funktionen nicht gekapselt werden und somit für komplexere Integration mehr Daten ausgetauscht werden müssen als bei webbasiertem Format. Der Autor bewertet daher den Durchsatz bei Change-Data-Capture mit der Wertzahl 3 (*stark*).

Latenz: sehr stark

Da Änderungen sofort übertragen werden ist eine Integration von Daten in Echtzeit möglich. Der Autor bewertet daher die Latenz bei Change-Data-Capture mit der Wertzahl 4 (*sehr stark*).

5.3.6. Database-Connection

Merkmale: tabellenbasierte pull-Integration

Robustheit: schwach

Consumer sind jederzeit von der Verfügbarkeit des Providers anhängig. Obwohl Datenbankverbindungen robust konzipiert sind, kann ein aktueller Datensatz nur dann garantiert werden, wenn alle Komponenten entlang des Integrationspfades verfügbar sind. Der Autor bewertet daher die Robustheit bei Database-Connection mit der Wertzahl 2 (*schwach*).

Integrität: sehr stark

Der integrierte Datensatz ist immer integer, wenn alle benötigten Daten in ihrer Gesamtheit vom Consumer periodisch, oder On-Demand angefragt werden. Wenn es eine klare Quell- und Zielbeziehung gibt, kommt es nicht zu Datenkonflikten, da das lesende System jederzeit Zugriff auf die aktuellen Daten hat. Der Autor bewertet daher die Integrität bei Database-Connection mit der Wertzahl 4 (*sehr stark*).

Kontrolle: schwach

Da Consumer jederzeit Zugriff auf die Datenstrukturen, sowie die vollen Datensätze haben, können weder Zeitpunkt noch Umfang der Integration vom Provider gesteuert werden. Es ist lediglich möglich den Umfang der freigegebenen Tabellen zu steuern. Der Autor bewertet daher die Kontrolle bei Database-Connection mit der Wertzahl 2 (*schwach*).

Flexibilität: schwach

Der Datenaustausch erfolgt direkt auf der Datenbankebene und wird nicht abstrahiert. Obwohl Verbindungsprotokolle standardisiert sind, können Treiber und Abfragedetails teilweise System- oder Technologiespezifisch stark abweichen. Technische, strukturelle und konzeptionelle Änderungen sind somit aufwändig und bedürfen einer umfassenden Anpassung der implementierten Integration. Der Autor bewertet daher die Flexibilität bei Database-Connection mit der Wertzahl 2 (*schwach*).

Durchsatz: stark

Bei tabellenbasierten Integrationen können protokollbedingt große Datenmengen innerhalb kurzer Zeit ausgetauscht werden. Die Ausführungszeit wird hierbei jedoch auch von der Performance des Datenbanksystems bestimmt, welches für die Bearbeitung umfangreicher Abfragen entsprechende Zeit benötigt. Der Autor bewertet daher den Durchsatz bei Database-Connection mit der Wertzahl 3 (*stark*).

Latenz: sehr stark

Mit Database-Connections können Daten in Echtzeit angefragt und sowohl virtuell als auch materialisiert integriert. Der Autor bewertet daher die Latenz bei Database-Connection mit der Wertzahl 4 (*sehr stark*).

5.3.7. Wrapper/Mediator

Merkmal: Systemnativität der Übertragungseinrichtung ist extern

Robustheit: sehr schwach

Wenn bei bestehenden Systemen bestimmte Integrationsmethoden nativ nicht zur Verfügung stehen, so können diese unter Umständen trotzdem in Betracht gezogen werden. Wie in Abschnitt 3.4.7 beschrieben, können API-Wrapper, oder Mediator eingesetzt werden, um APIs systemextern zu ertüchtigen und zu erweitern. So kann die Bewertung im Einzelfall entsprechend den technischen Rahmenbedingungen angepasst werden. Native APIs sind grundsätzlich als Robuster zu bewerten, als extern implementierte Datenübertragungseinrichtungen. Jedoch bietet sich hier die Möglichkeit nicht vorhandene Funktionalität nicht von vornherein bei der Designentscheidung mit einzubeziehen. Ist beispielsweise eine Implementierung mit Webhooks sinnvoll, jedoch vom zu integrierenden System nicht vorgesehen, so kann mithilfe einer Middleware ein Wrapper implementiert werden, welcher die native CDC-Funktionalität des Provider-Systems nutzt und die Entsprechenden Daten über Webhooks an die Consumer-Systeme verteilt. Je nach Anwendungsfall kann zum Beispiel auch ein File-Import mithilfe einer Database-Connection oder mit Webservices automatisiert werden, so dass gleichzeitig ein hoher Durchsatz erreicht wird, die Implementierung jedoch auch kontrollierbar und flexibel bleibt.

Um dem oben beschriebenen Sachverhalt Rechnung zu tragen, wird bei einer nicht-nativen Implementierung der entsprechenden Methode (API-Wrapper/Mediator) die Robustheit mit der Wertzahl 1 *sehr schwach* bewertet.

5.4. Zusammenfassung der Bewertungsergebnisse

In diesem Abschnitt werden die Erkenntnisse aus diesem Kapitel zusammengefasst, um die Forschungsfrage FF 3 (*Wie lassen sich Integrationsarchitekturen in Bezug auf die individuellen Anforderungen modellbasiert bewerten?*) abschließend zu beantworten.

Zunächst werden die Ergebnisse der Gegenüberstellung beschrieben und zusammengefasst. Diese bildet die Datengrundlage für die initiale Parametrisierung des GAP-Modells. An dieser Stelle ist hervorzuheben, dass die Klassifizierung und Bewertung sowohl von Integrationstopologien und -paradigmen technologieagnostisch erfolgt ist, auch wenn bestimmte Technologien für die jeweiligen Implementierungen prädestiniert sind. So liegt bei eventuell Webservices eine Implementierung als Service-Driven-Integration in einem Integration-Mesh nahe, jedoch können Webservices auch für eine Storage-Driven-Integration im Integration-Hub genutzt werden. Letztendlich sind die drei Architekturebenen trennscharf zu bewerten.

5.4.1. Integrationstopologien

Wie oben beschrieben, wurden die vier Integrationstopologien *Integration-Hub*, *-Bus*, *-Tree* und *-Mesh* in Bezug auf *Robustheit*, *Integrität*, *Skalierbarkeit*, *Kontrolle* und *Flexibilität* bewertet. Die hier ermittelten Bewertungsergebnisse bilden die Grundlage für die Anwendung des in Kapitel 4 beschriebenen GAP-Modells zur Identifikation einer geeigneten Integrationstopologie. Abbildung 39 zeigt eine graphische Darstellung der Ergebnisse als Netzdiagramme.

Integration-Hub ist dann eine gute Wahl, wenn hohe Anforderungen an die Kontrolle über die Integrationslandschaft gestellt werden und Integrität und Flexibilität wichtig sind. Der stark zentralisierte Ansatz geht dabei jedoch zu Lasten der Robustheit und Skalierbarkeit, da der Integration-Hub sowohl Single-Point-of-Failure als auch Flaschenhals für alle Integrationen ist.

Integration-Bus ist eine ausgewogene Lösung und ermöglicht eine sehr Flexible Integrationslandschaft. Ein Problem an der zentralen „Datenautobahn“ ist, dass ihr Ausfall alle Integrationen zum Stillstand bringt (Robustheit). Dafür können jedoch jederzeit Systeme an- und abgekoppelt werden (Flexibilität und Skalierbarkeit). Auch ist durch die Kanalisierung des gesamten Informationsaustausches jederzeit die Kontrolle über die Kommunikation gewährleistet.

Integration-Tree ist eine gute Lösung, wenn Integrationslandschaften nicht flexibel angepasst werden müssen, jedoch Robustheit und Integrität hohe Wichtigkeit haben. Vor allem ist der Integration-Tree jedoch sehr skalierbar, solange die ursprüngliche Konzeption der Integrationspfade nicht verändert werden muss.

Integration-Mesh ist als Integrationstopografie besonders geeignet, wenn Integrationslandschaften sehr flexibel, robust und skalierbar sein müssen. Dies geht jedoch stakt zu Lasten der Kontrolle und Integrität im Konnektivitätsökosystems. Das Netzwerk kann beliebig angepasst und verändert werden, führt jedoch ohne entsprechende Maßnahmen zu unkontrollierbarem Wildwuchs.

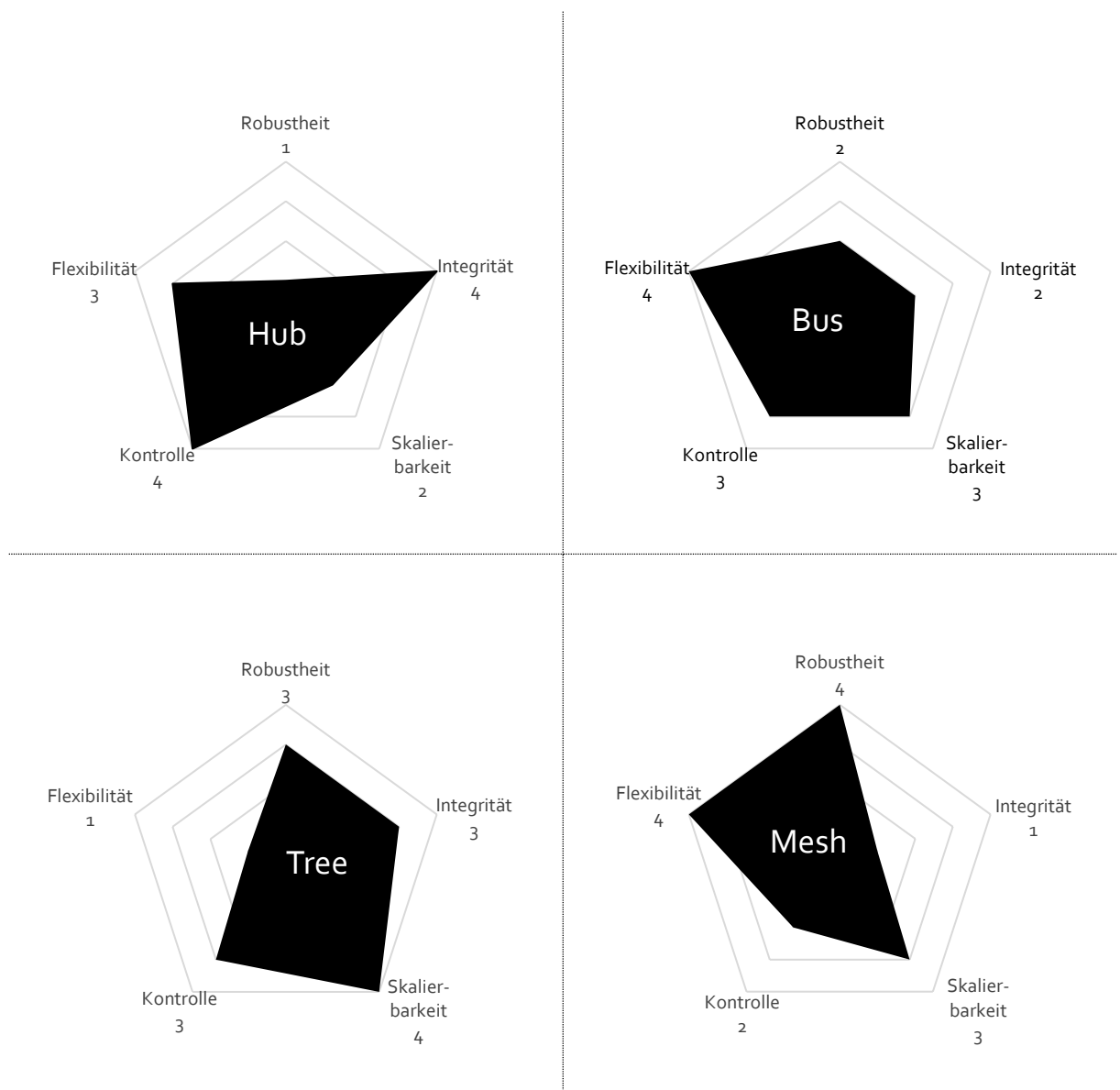


Abbildung 39: Gegenüberstellung ausgewählter Integrationstopologien

Bei der Betrachtung der Bewertungsergebnisse wird klar, dass keine Lösung in allen Dimensionen besonders stark, oder schwach ist. Hierbei ist ein interessanter Aspekt, dass die Summe aller Bewertungspunkte bei allen vier bewerteten Topologien gleich ist. Dies war vom Autor nicht geplant und ergab sich erst in der Auswertung der. Da die Merkmale der jeweiligen Topologien qualitativ im Vergleich mit den anderen Topologien bewertet wurden, ist dies ein klares Zeichen dafür, dass ohne entsprechende Priorisierung der Merkmale keine Topologie im Vorteil ist. Somit bestätigt sich die Annahme, dass für die Auswahl der richtigen Topologie eine Gewichtung der Anforderungen von Nöten ist. Dabei ist hervorzuheben, dass in einer Integrationslandschaft mehrere Topologien miteinander kombiniert werden können. So kann ein Teil der Integrationen über einen Integration-Hub abgewickelt werden, wo ein anderer Teil der Systeme über Direktverbindungen in einem Integration-Mesh angebunden wurde. Damit lassen sich situativ Vorteile beider Topologien für ein zweckdienliches Ergebnis kombinieren.

Details zur Auswertung der auf die vergleichende Bewertung der betrachteten Integrationstopologien findet sich in Anhang 8 ab Seite 155.

5.4.2. Integrationsparadigmen

Wie oben beschrieben, wurden die drei Integrationsparadigmen *Storage-*, *Service-* und *Event-Driven-Integration* in Bezug auf *Robustheit*, *Integrität*, *Kontrolle*, *Flexibilität*, *Durchsatz* und *Latenz* bewertet. Die hier ermittelten Bewertungsergebnisse bilden die Grundlage für die Anwendung des in Kapitel 4 beschriebenen GAP-Modells zur Identifikation eines geeigneten Integrationsparadigmas. Abbildung 40Abbildung 20 zeigt eine graphische Darstellung der Ergebnisse als Netzdiagramme.

Storage-Driven-Integrations sind besonders geeignet, wenn große Datenmengen integriert und zuverlässig vorgehalten werden müssen und Aktualität und Synchronität der Daten nicht entscheidend sind. Dem Paradigma einer Storage-Driven-Integration folgend lassen sich Integrationen aufgrund der Materialisierung der Daten sehr robust implementieren. Ferner kann je nach Implementierung ein hoher Durchsatz und gute Kontrolle erreicht werden. Dabei gibt es jedoch große Schwächen bei der Flexibilität, sowie bei der Latenz und Integrität der Daten.

Service-Driven-Integrations sind eine gute Wahl, wenn die Systemlandschaft flexibel gestaltet werden soll und Systeme punktgenau in Echtzeit miteinander interagieren sollen. Aufgrund der Virtualisierung können Integrationen mit geringer Latenz und hoher Integrität implementiert werden. Dies geht jedoch zu Lasten von Robustheit, Kontrolle und Durchsatz. Es können somit sehr dynamische Integrationslösungen gestaltet werden, wobei auf die Zuverlässigkeit der jeweiligen Akteure besonders Acht gegeben werden muss.

Event-Driven-Integrations sind besonders für Szenarien geeignet, in denen große Datenmengen in Echtzeit zwischen einer Vielzahl von Systemen synchronisiert werden sollen. Die Verbindung zwischen Systemen kann mit Event-Driven-Integrations können sehr flexibel gestaltet werden. Da die Kommunikation unmittelbar durch eine Änderung der Informationen im Provider-System initiiert wird (push), kann eine sehr geringe Latenz und ein starker Durchsatz erreicht werden. Nachteilig sind eine geringe Kontrolle und verhältnismäßig schwache Robustheit der Integrationen.

Im direkten Vergleich zeigt sich, dass alle drei Integrationsparadigmen unterschiedliche Stärken und Schwächen haben. Interessant hierbei ist, dass Service-Driven-Integration in der Bewertung beinahe diametral zur Storage-Driven-Integration steht, wohingegen Event-Driven-Integration insgesamt ausgeglichen bewertet ist.

Details zur Auswertung der auf die vergleichende Bewertung der betrachteten Integrationsparadigmen findet sich in Anhang 9 ab Seite 156.

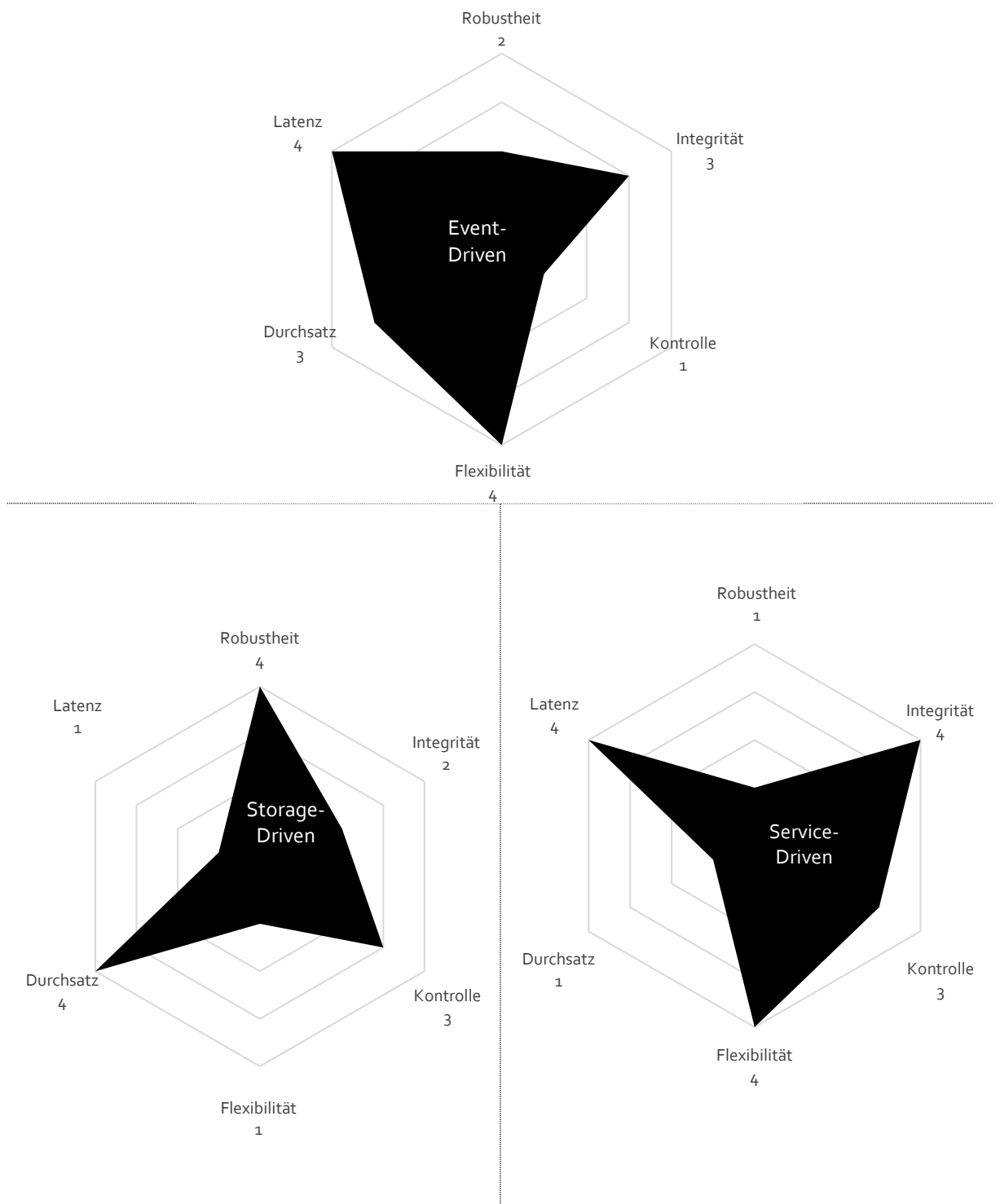


Abbildung 40: Gegenüberstellung ausgewählter Integrationsparadigmen

5.4.3. Integrationsmethoden

Wie oben beschrieben, wurden die sechs Integrationsmethoden File-Exports, File-Imports, Webhooks, Webservices, Change-Data-Capture und Database-Connection in Bezug auf Robustheit, Integrität, Kontrolle, Flexibilität, Durchsatz und Latenz bewertet. Die in Abbildung 41 dargestellten Bewertungsergebnisse können im Rahmen des in Kapitel 4 beschriebenen Modells angewendet werden, um für einen bestimmten Anwendungsfall geeignete Integrationsmethoden zu identifizieren.

File-Exports sind eine gute Integrationsmethode, wenn große in sich integre Datenmengen in kurzer Zeit möglichst robust aus einem System exportiert werden sollen und die Kontrolle über die weitere Nutzung dieser Daten und ihre Aktualität im weiteren Verlauf weniger wichtig sind.

File-Imports eignen sich gut, wenn große Mengen vertrauenswürdiger Daten möglichst schnell und robust in ein System geladen werden sollen, ohne dass durch die Integrationsmethode selbst sichergestellt werden muss, dass die Daten korrekt und aktuell sind.

Webhooks sind eine gute Methode für flexible und hoch kontrollierte Echtzeitintegration von Daten und Services. Diese Integrationsmethode hat dabei eine vergleichsweise geringe Robustheit und ist nicht gut für den einmaligen Austausch großer Datenmengen innerhalb einer begrenzten Zeit geeignet. Stattdessen sollte sie im Kontext eines kontinuierlichen Datenflusses eingesetzt werden.

Webservices eignen sich besonders für den kontrollierten, zielgenauen und unmittelbaren Zugriff auf Daten und Services eines Provider-Systems in Echtzeit. Diese Integrationsmethode ist dabei weniger geeignet für den sehr Robusten Austausch großer Datenmengen und sollte eher im Kontext einer engen funktionalen Kopplung von Systemen eingesetzt werden.

Change-Data-Capture bietet eine ausgewogene Möglichkeit, um Systeme auf Datenbankebene in Echtzeit miteinander zu synchronisieren, wobei diese Methode aufgrund der fehlenden Abstraktionsschicht weniger flexibel ist. Ähnlich wie bei Webhooks, ist diese Integrationsmethode vor allem im Kontext eines kontinuierlichen Datenflusses im Konnektivitätsökosystem zu sehen.

Database-Connection ist als Integrationsmethode gut geeignet, wenn auf die aktuelle Datenbasis integrierter Systeme zugegriffen werden soll. In Eingeschränkter Weise können so auch Transaktionen bzw. Services integriert werden, wobei diese Integrationen bedingt durch die fehlende Abstraktion und Kapselung weniger robust, relativ unflexibel und nicht einfach kontrollierbar sind.

Wie in Abbildung 41 dargestellt, zeigt sich im direkten Vergleich der verschiedenen Integrationsmethoden, dass dateibasierte Integrationen bei hohem Durchsatz eine hohe Robustheit erreichen, wohingegen webbasierte Integrationen eine gute Integrität, Flexibilität Kontrolle und Latenz ermöglichen. Tabellenbasierte Integrationen wiederum erscheinen in Bezug auf Stärken und Schwächen am ausgeglichensten.

Details zur Auswertung der auf die vergleichende Bewertung der betrachteten Integrationsmethoden findet sich in Anhang 10 ab Seite 157.

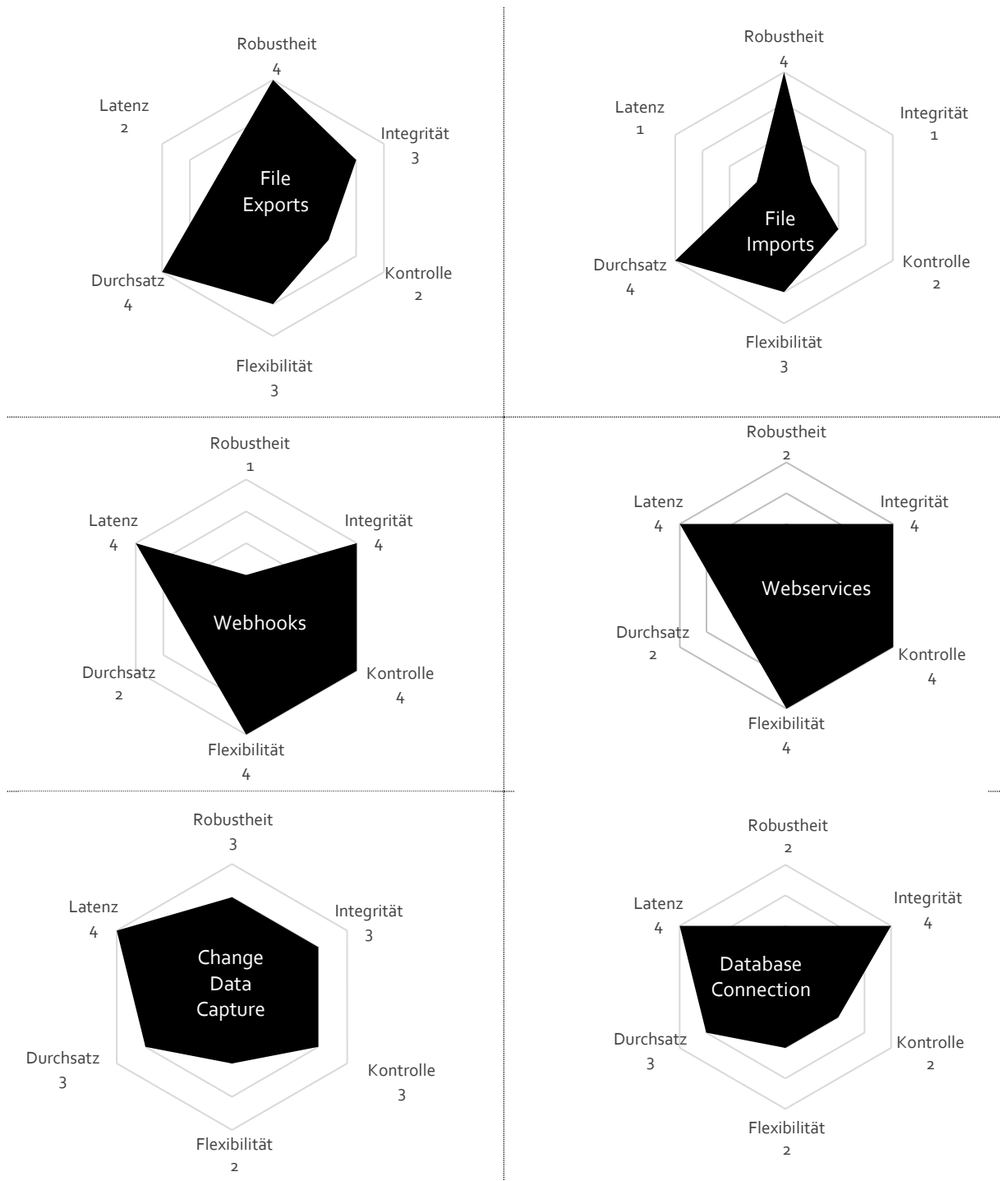


Abbildung 41: Gegenüberstellung ausgewählter Integrationsmethoden

6. Empirische Validierung des GAP-Modells anhand von Fallstudien

In diesem Kapitel erfolgt die Validierung des GAP-Modells anhand dreier Fallstudien. Dabei wird das GAP-Modell jeweils auf tatsächlich implementierte Anwendungsfälle angewendet und anhand unabhängig vorliegender Bewertungsergebnisse validiert.

6.1. Versuchsaufbau

Für die Validierung des GAP-Modells wurden drei Anwendungsfälle mit unterschiedlichen Anforderungen und Implementierungen ausgewählt. Wie in Abbildung 42 dargestellt, wird das GAP-Modell auf jeden der drei Anwendungsfälle angewendet und die Empfehlungen mit den tatsächlich implementierten Integrationsarchitekturen verglichen. Die Ergebnisse des GAP-Modells werden anschließend anhand der Ergebnisse eines unabhängig erfolgten Application Assessments nach TOGAF verifiziert. Das GAP-Modell wird dann als validiert angesehen, wenn die Bewertungsergebnisse mit den Beobachtungen und Ergebnissen des Application Assessments übereinstimmen.

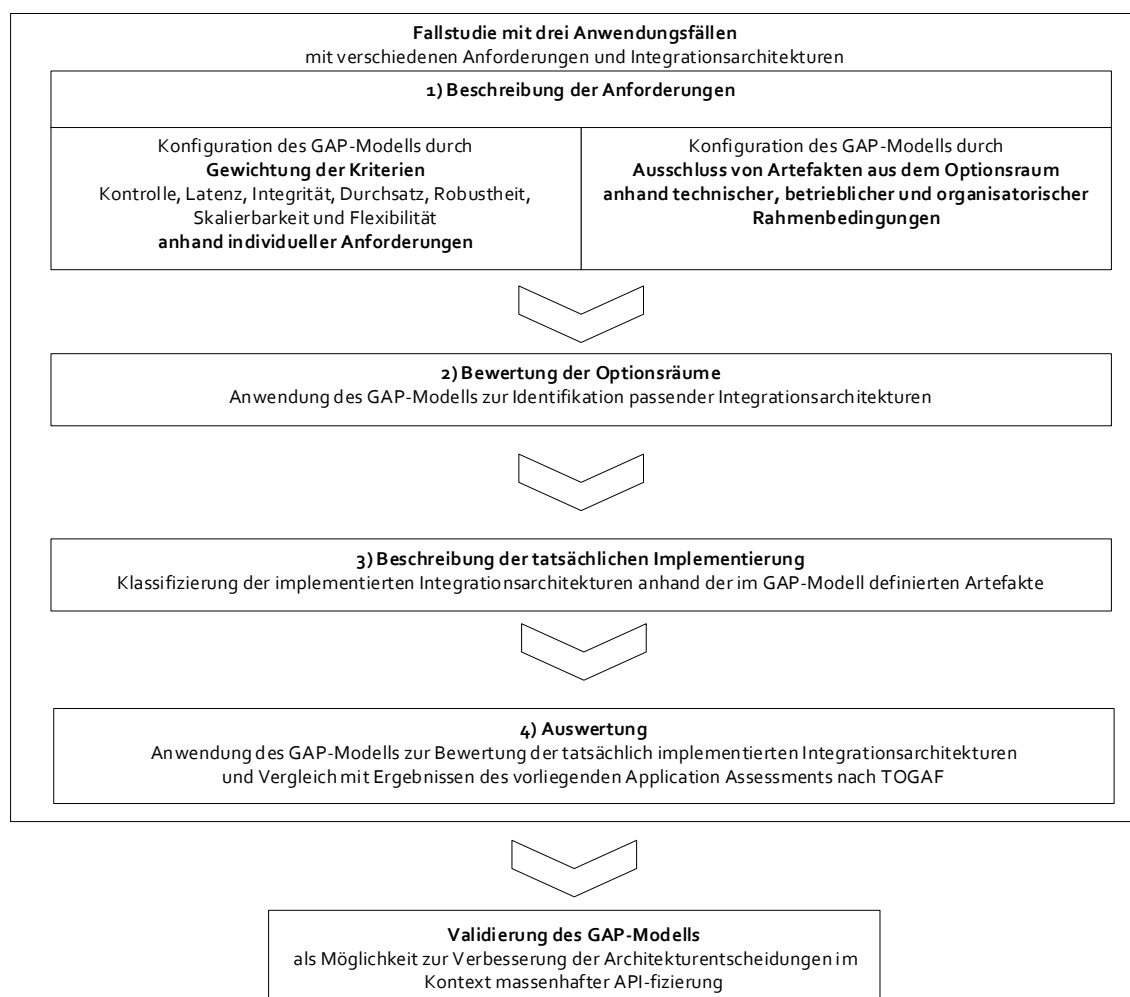


Abbildung 42: Versuchsaufbau zur empirischen Validierung des GAP-Modells

6.1.1. Vorgehensweise

Zur empirischen Validierung des GAP-Modells wird dieses wie oben beschrieben in drei ausgewählten Fallstudien angewendet. Im Anschluss erfolgt eine Auswertung der Ergebnisse zur konzeptionellen und inhaltlichen Validierung des GAP-Modells. Das Vorgehen gliedert sich dabei pro Fallstudie in folgende Schritte:

Beschreibung der Anforderungen

Im ersten Schritt wird der Anwendungsfall umrissen und die Anforderungen definiert. Ziel ist die Konfiguration des GAP-Modells. Das Anforderungsprofil wird dabei anhand der Gewichtung entlang der in Kapitel 4 definierten Dimensionen aufgebaut. Zusätzlich werden technische, betriebliche und organisatorische Rahmenbedingungen erfasst, welche bestimmte Artefakte ausschließen und den Optionsraum somit einschränken.

Bewertung der Optionsräume

Im zweiten Schritt wird das GAP-Modell entsprechend der im ersten Schritt erfolgten Konfiguration ausgeführt. Die Optionsräume werden automatisch generiert und bewertet. Das GAP-Modell identifiziert dabei alle Integrationsarchitekturen, welche den Anforderungen gerecht werden. Hieraus ergibt sich die Architekturempfehlung für den jeweiligen Anwendungsfall.

Beschreibung der tatsächlichen Implementierung

Im dritten Schritt wird auf die tatsächlich implementierten Integrationsarchitekturen eingegangen. Dabei erfolgt eine Klassifizierung anhand der in Kapitel 3 definierten Taxonomie, sowie eine Beschreibung der beobachteten Vor- und Nachteile der implementierten Integrationen.

Vergleich mit den Ergebnissen des Application Assessments

Im letzten Schritt wird das GAP-Modell zur Bewertung der tatsächlich implementierten Integrationsarchitekturen angewendet und es erfolgt ein Vergleich mit den Ergebnissen eines unabhängig durchgeführten Application Assessments. Die in diesem Kontext genutzten Bewertungskriterien sind unten in Abschnitt 6.1.2 beschrieben. Hierbei ist zu beachten, dass das GAP-Modell die Integration zwischen Systemen betrachtet, wobei das Application Assessment die Anforderungserfüllung des Systems selbst beschreibt. Eine direkte Übersetzung von den Bewertungskriterien des GAP-Modells in die Kriterien des Application Assessments ist daher nicht möglich. Die Auswertung erfolgt jedoch anhand einer Kontextualisierung und qualitativen Einordnung der Bewertungsergebnisse.

6.1.2. Kriterien des Application Assessments

Die betrachteten Anwendungsfälle wurden in einem Application Assessments nach TOGAF anhand der folgenden Kategorien bewertet. Die unten beschriebenen Kategorien lassen sich mit Dimensionen des GAP-Modells in Verbindung setzen. Darüber hinaus werden auch Kategorien, wie beispielweise der „Business Fit“ bewertet, welche jedoch für die Betrachtung von Systemintegrationen unerheblich sind und somit keine weitere Beachtung finden.

API First

Das Kriterium „API First“ beschreibt, wie gut die derzeitige Implementierung die Vision eines modernen Konnektivitätsökosystems umsetzt. Es ist somit ein klarer Hinweis auf die Güte der historischen Designentscheidungen in Bezug auf verfügbare Integrationsmethoden. Im Kontext einer Cloud-orientierten IT-Strategie sind insbesondere webbasierte Integrationsmethoden wünschenswert (siehe auch „Hosting Type“). Die Bewertung erfolgt auf folgender Ordinalskala mit fünf Ausprägungen. Diese findet auch bei der anderen Kriterien Anwendung.

- *Major Gap in Coverage* (Anforderungen weit verfehlt)
- *Minor Gap in Coverage* (Anforderungen verfehlt)
- *Near-Complete Coverage* (Anforderungen nahezu erfüllt)
- *Complete Coverage / Satisfactory* (Anforderungen erfüllt)
- *Outstanding / Industry Lead* (Anforderungen übererfüllt)

Aus Integrationssicht bedeutet eine schlechte Bewertung, dass Schnittstellen nicht, oder nur eingeschränkt angeboten werden. Eine gute Bewertung erfolgt, wenn alle relevanten Daten und Funktionalitäten über hochwertige APIs zur Verfügung gestellt werden. API-First müsste sich somit mittelbar auf die Kriterien *Flexibilität*, *Skalierbarkeit*, *Latenz* und *Integrität* im GAP-Modell auswirken.

Modular Architecture

Das Kriterium „Modular Architecture“ beschreibt, inwieweit die bewertete Applikation modularisiert ist und über Möglichkeiten zur Kapselung von Daten und Funktionalität verfügt. Die Bewertung erfolgt auf derselben Ordinalskala, die API-first (siehe oben). Eine schlechte Bewertung bedeutet aus Integrationssicht, dass Daten und Funktionalitäten nicht modularisiert und gekapselt sind, so dass sich dies voraussichtlich negativ auf *Flexibilität*, *Kontrolle* und *Skalierbarkeit* im Sinne des GAP-Modells auswirkt.

Security and Compliance

Das Kriterium „Security and Compliance“ beschreibt, inwiefern die bestehende Applikation die Sicherheitsanforderungen erfüllt. Für die Bewertung kommt dieselbe Ordinalskala zum Einsatz, wie bei „API-First“ (siehe oben). Eine schlechte Bewertung in Bezug auf Security and Compliance müsste sich im GAP-Modell an einer Schwäche bei *Kontrolle* und *Integrität* abzeichnen.

Operational Excellence

Das Kriterium „Operational Excellence“ Beschreibt die Qualität des Applikationsbetriebs, bzw. inwieweit die Applikation mit vertretbarem Aufwand entsprechend der Anforderungen betrieben werden kann. Dies ist insbesondere im Bereich der validierten Computersysteme relevant, da ein guter Betrieb für die operative Integrität zwingend notwendig ist. Für die Bewertung kommt dieselbe Ordinalskala zum Einsatz, wie bei „API-First“ (siehe oben).

Eine schlechte Bewertung in Operational Excellence müsste im GAP-Modell mit einer schwachen Bewertung von *Robustheit*, *Kontrolle* oder *Integrität* korrespondieren.

Hosting Type

Der „Hosting Type“ beschreibt die Laufzeitumgebung der Applikation. Entsprechend der IT-Strategie werden im betrachteten Anwendungsbeispiel Cloud-Lösungen präferiert. Hosting Types werden im vorliegenden Application Assessment wie folgt bewertet. Im Kontext einer Cloud-first IT-Strategie ist im betrachteten Anwendungsfall die Bewertungsskala als Ordinalskala mit Rangordnung ausgeprägt.

- *On-prem (nicht akzeptabel)*: vom Unternehmen auf unternehmenseigenen Servern betriebene Applikationen
- *Hybrid (zu vermeiden)*: vom Unternehmen teilweise auf unternehmenseigenen Servern und teilweise auf Cloud-Infrastruktur betriebene Applikationen.
- *Private Cloud (akzeptabel)*: vom Unternehmen auf besonders geschützter Cloud-Infrastruktur betriebene Applikationen
- *Public Cloud (gut)*: vom Unternehmen auf öffentlich zugänglicher Cloud-Infrastruktur eines Dienstleisters betriebene Applikationen
- *SaaS (sehr gut)*: vom Dienstleister betriebene Applikationen

In Bezug auf Integrationen hat der Hosting Type Auswirkungen auf Topologie und Methode der möglichen Implementierungen. So kann beispielsweise eine vom Unternehmen auf unternehmenseigenen Servern betriebene Applikation (on-prem) nicht ohne Weiteres mit einer Public-Cloud oder SaaS-Lösung kommunizieren. Hierfür wird beispielsweise ein API-Gateway benötigt, was wiederum zulasten der Robustheit geht. Auch können bestimmte Integrationsmethoden ohne zusätzlichen Aufwand nur innerhalb eines gesicherten Netzwerkes (on-prem, Hybrid, Private Cloud) wirklich sicher betrieben werden. Ein ungeeigneter Hosting-Type müsste sich demnach im GAP-Modell negativ auf das Kriterium Kontrolle auswirken.

Technical Fit

Das Kriterium des „Technical Fit“ wird sowohl für das bestehende Anforderungsprofil („Technical Fit for now“), als auch für die Zukunft („Technical Fit for future“) betrachtet. Hierbei wird bewertet, wie gut die derzeitige Implementierung den derzeitigen Anforderungen entspricht und ob sich die Applikation im weiteren Lebenszyklus an voraussichtliche Änderungen anpassen lässt. Die Bewertung erfolgt auf einer dreigliedrigen Ordinalskala von *Low (niedrig)*, über *Medium (mittel)*, nach *High (hoch)*.

Eine schlechte Bewertung beim Technical Fit könnte beim GAP-Modell an einer schlechten Abdeckung der Anforderungen insbesondere beim Kriterium Durchsatz, Flexibilität und Robustheit sichtbar werden. Ein niedriger „Technical fit for future“ könnte zudem auf eine geringe Skalierbarkeit hindeuten.

Off-the-Shelf

Das Kriterium „Off the shelf“ beschreibt, ob es sich um ein Standardprodukt handelt, uns wie weit dieses verändert und erweitert wurde. Für die Bewertung kommt dieselbe Ordinalskala zum Einsatz, wie bei „API-First“ (siehe oben).

Im betrachteten Anwendungsfall werden Standardlösungen grundsätzlich bevorzugt, da diese in Bezug auf Entwicklungs- und Betriebskosten tendenziell am günstigsten sind. Eine schlechte Bewertung kann im Kontext von Integrationsarchitekturen beispielsweise bedeuten, dass Integrationsmöglichkeiten nicht nativ implementiert sind und stattdessen API-Wrapper oder Mediatoren zum Einsatz kommen. Eine direkte Verbindung mit den Bewertungen im GAP-Modell kann für den Merkmal der *Systemnativität der Übertragungseinrichtung* hergestellt werden, so dass eine schlechte Bewertung in dieser Kategorie mit einer schwachen Robustheit korrespondieren könnte.

6.2. Anbindung verschiedener Systeme an ein operatives Data Warehouse

In folgendem wird das GAP-Modell für den Anwendungsfall eines klassischen Data Warehouse angewendet. Vorgehen und Ergebnisse dieser Fallstudie 1 sind in Abbildung 43 dargestellt. Ausgehend vom Anforderungsprofil wird der Optionsraum aufgebaut und bewertet. Aus den möglichen Designartefakten ergeben sich 48 mögliche Archetypen für Integrationsarchitekturen. Aus diesen können fünf Archetypen für geeignete Integrationsarchitekturen identifiziert werden. Dabei wird als beste Lösung vorgeschlagen, dass sich das operative Data Warehouse die zu integrierende Daten über Webservices direkt aus den Quellsystemen ausliest und zentral vorhält. Die Detaillierten Tabellen und Darstellungen zur Anwendung des GAP-Modells für ein operatives Data Warehouse finden sich in Anhang 11 ab Seite 158.

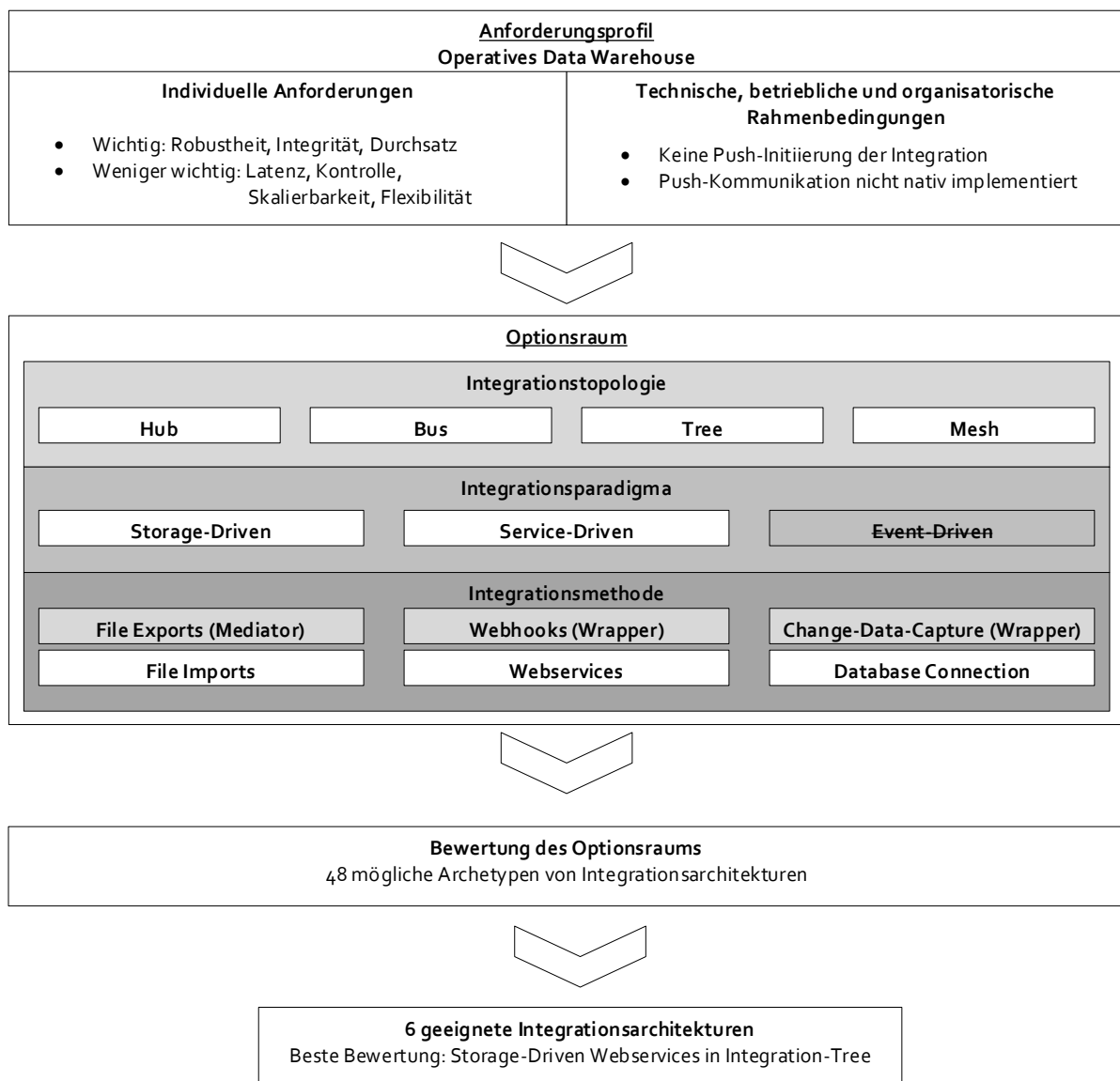


Abbildung 43: Vorgehen und Ergebnisse von Fallstudie 1

6.2.1. Beschreibung der Anforderungen

In Folgendem werden die Anforderungen an ein operatives Data Warehouse in Bezug auf die Integrationsarchitektur im betrachteten Anwendungsfall beschrieben.

Nutzungszweck des operativen Data Warehouse ist es, Daten aus verschiedenen operativen Systemen an einem Ort für Dashboards und Reporting zur Verfügung zu stellen. Anwendungsbereich ist sowohl die operative Prozesssteuerung als auch die Nutzung für kurzfristige Geschäftsentscheidungen. Die Daten sollten dabei an einem zentralen Ort zusammengezogen und unabhängig vom Status der Quellsysteme verfügbar sein. Die Belastung der operativen Quellsysteme soll dabei so gering wie möglich sein. Eine Tagesaktuelle Darstellung der Daten ist ausreichend, so dass keine Echtzeitintegration benötigt wird. Abbildung 44 zeigt das individuelle Anforderungsprofil entsprechend der gewichteten Kriterien.

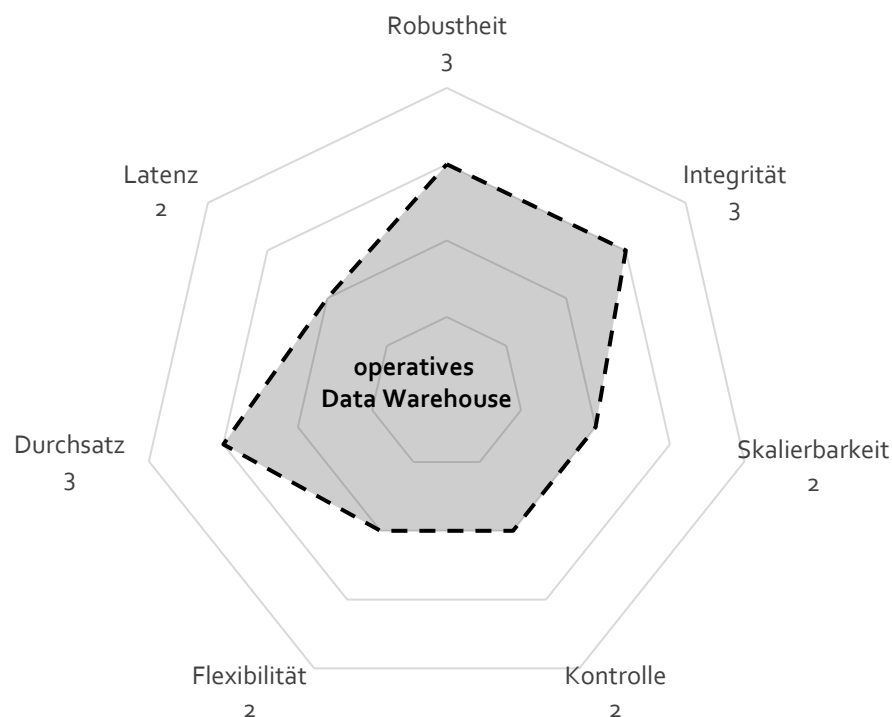


Abbildung 44: Anforderungsprofil an Integrationen für ein operatives Data Warehouse

Im Fokus der Anforderungen stehen die Kriterien *Durchsatz*, *Robustheit* und *Integrität*. Geeignete Integrationsarchitekturen sollten daher in diesem Punkt mindestens mit der Wertzahl 3 (*stark*) bewertet sein.

Die Anforderung an einen sehr guten Durchsatz der Integrationen ergibt sich aus der Überlegung, dass die Last auf den operativen Systemen möglichst konstant gehalten werden soll, um die operative Performance dieser nicht zu beeinträchtigen. In einem international agierenden Unternehmen werden die angebundenen Systeme ununterbrochen genutzt, so dass die operative Integrität gefährdende Lastspitzen möglichst vermieden werden sollten. Bei lokalen Nutzungsprofilen wäre diese Anforderung weniger wichtig, da sich Integrationsprozesse in vorhersehbare Zeiten niedrigerer Last, beispielsweise als nächtlicher Job, einplanen lassen.

Robuste Integrationen werden benötigt, um zu gewährleisten, dass die benötigten Informationen im operativen Data Warehouse immer zur Verfügung stehen, auch wenn die angebundenen Quellsysteme kurzfristig nicht verfügbar sind. Die Anforderungen nach starker Integrität der

Daten beschreibt den Anspruch, dass die verfügbaren Daten zu jeder Zeit korrekt und konfliktfrei sind. Nur so können datengetriebene Entscheidungen souverän getroffen werden, ohne dass Zweifel an der Richtigkeit der gesammelten Daten besteht.

Die Aspekte der *Latenz*, *Skalierbarkeit*, *Kontrolle* und *Flexibilität* sind im betrachteten Kontext weniger wichtig. In diesen Aspekten ist bei geeigneten Integrationsarchitekturen eine Bewertung mit der Wertzahl 2 (*schwach*) akzeptabel.

Eine schwache Latenz ist akzeptabel, da Daten nur Tagesaktuell vorgehalten werden müssen und somit ein gewisser Zeitverzug in Kauf genommen werden kann. Eine schwächere Kontrolle ist akzeptabel, da lediglich Daten aus Quellsystemen eingelesen und nicht manipuliert werden.

Flexibilität und Skalierbarkeit spielen keine übergeordnete Rolle, da keine unplanbare Änderung am umgebenden Konnektivitätsökosystem zu erwarten sind. Nichtsdestotrotz dürfen diese Punkte nicht gänzlich vernachlässigt werden, um nachhaltigen Betrieb und kontinuierliche Weiterentwicklung nicht zu gefährden.

Zusätzlich zu diesen Anforderungen bestehen technische Einschränkungen, die den möglichen Optionsraum einengen. Wie in Abbildung 43 dargestellt entfällt das Event-Driven-Paradigma, da die angebundenen Systeme größtenteils nicht über die Fähigkeit zur proaktiven Initiierung der Kommunikation verfügen und keine Änderungen an den Quellsystemen vorgenommen werden soll. Zudem sind Integrationsmethoden mit push-Kommunikation nicht nativ in den anzubindenden Systemen implementiert, so dass für File-Exports, Webhooks und Change-Data-Capture ein API-Wrapper, bzw. Mediator implementiert werden müsste.

6.2.2. Anwendung des GAP-Modells

Unter Anwendung des GAP-Modells wurden automatisch entsprechend der in Kapitel 3 definierten Klassifizierung von Designartefakten alle möglichen Architekturausprägungen miteinander bewertet. Eine detaillierte Darstellung des gesamten Optionsraums findet sich in Anhang 11 ab Seite 158.

Für das definierte Anforderungsprofil ermittelt das GAP-Modell sechs passende Integrationsarchitekturen, welche das Anforderungsprofil vollständig erfüllen. Diese sind in Tabelle 13 dargestellt. Sie zeigt Wertpunkte und Bewertung der sieben Architekturarchetypen, welche die engere Auswahl für die Zielarchitektur ausmachen. Die Zeilen sind dabei absteigend nach Bewertungsergebnis sortiert.

Alle Optionen entsprechen genau den hohen Mindestanforderungen an Robustheit und Integrität. Die geringe Anforderung an Skalierbarkeit wird von allen Optionen übererfüllt. Auch Kontrolle bzw. Latenz sind bei fünf aus sechs Optionen höher als erforderlich, wobei hier eine gewisse Wechselwirkung zu erkennen ist. Bei der Flexibilität und Durchsatz können nur zwei von sechs Optionen eine höhere Bewertung aufzeigen.

Das GAP-Modell macht eine klare Empfehlung zu Storage-Driven-Integration in einer Hub- oder Tree-Topologie und lässt dabei die Wahl der Integrationsmethode relativ uneingeschränkt. Auch eine dateibasierte Service-Driven Integration ist denkbar. Laut GAP-Modell werden die individuellen Anforderungen am besten mit Storage-Driven Webservices in einem Integration-Tree erreicht. Idealerweise fragt das operative Data Warehouse also in regelmäßigen Abständen über Webservices bei den Quellsystemen den aktuellen Datenstand ab und kopiert diesen physisch in seine Datenbank. Alternativ kann der Datenaustausch auch mittels eines Webhook-Wrappers, über direkten Datenbankzugriff, oder dateibasiert zweckdienlich realisiert werden.

				Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Anforderungsprofil				3	3	2	2	2	3	2	
Architekturarchetypen	Tree	Storage-Driven	Web-services	3	3	4	3	2	3	3	6,82
	Bus	Storage-Driven	Web-services	3	3	3	3	3	3	3	6,73
	Tree	Storage-Driven	Webhooks	3	3	4	3	2	3	3	6,65
	Bus	Storage-Driven	Database-Connection	3	3	3	3	2	4	3	6,38
	Tree	Service-Driven	File-Imports	3	3	4	3	2	3	3	6,24
	Tree	Storage-Driven	File-Exports	3	3	4	3	2	4	2	6,13

Tabelle 13: Bewertungsergebnisse des GAP-Modells für Integrationen mit einem operativen Data Warehouse

Bezüglich der Integrationstopologie werden sowohl Integration-Bus als auch Integration-Tree vorgeschlagen, wobei Hub und Mesh als unpassend eingeordnet werden. Grund hierfür sind die Schwächen in Bezug auf Robustheit, bzw. Integrität, welche die Zielvorgaben des Anforderungsprofils klar verfehlen. Somit kann die Wahl der Topologie die Integrationen in diesem Anwendungsfall nicht erheblich verbessern, die falsche Wahl jedoch verschlechtert die Zweckdienlichkeit deutlich.

In Bezug auf das Integrationsparadigma zeigt das GAP-Modell klar zum Storage-Driven-Paradigma. Durch das physische Kopieren der Daten von den Quellsystemen in das operative Data Warehouse kann insbesondere eine starke Robustheit gewährleistet werden. Dies spiegelt sich auch in der in Abschnitt 6.2.3 beschriebenen tatsächlichen Implementierung dar und ist typisch für ETL-Prozesse. Lediglich für die Integrationsmethode File-Imports schlägt das GAP-Modell das Paradigma Service-Driven-Integrations vor. Dabei würde das Auslesen der Daten als Datei in einen Systemservice eingebettet werden. So könnten zum Beispiel zusätzlich zur ausgetauschten Datei weitere Metadaten ausgetauscht werden. Zusätzlich müsste die Datei bereits vom Quellsystem so formatiert sein, dass sie vom operativen Data Warehouse ohne aufwändige Transformationen eingelesen werden kann. Dies wäre eine Antwort auf die Nachteile bei der Integrität von File-Imports bei gleichzeitiger Ausnutzung des starken Durchsatzes dieser Integrationsmethode.

Für die Integrationsmethode zeigen die Ergebnisse des GAP-Modells auf pull-Kommunikation basierende Lösungen als geeignet an. Bemerkenswert ist zudem, dass selbst nicht nativ implementierte Webhooks empfohlen werden, obwohl hierfür ein wenig robuster Wrapper implementiert werden müsste. Die schwache Robustheit wird durch die starke Robustheit des Storage-Driven-Paradigmas und der Tree-Topologie kompensiert. Dabei stärkt die Webhook-Methode insbesondere die Integrität der Integration.

Abbildung 45 zeigt die ausgewählten Optionen im Vergleich mit dem gegebenen Anforderungsprofil.

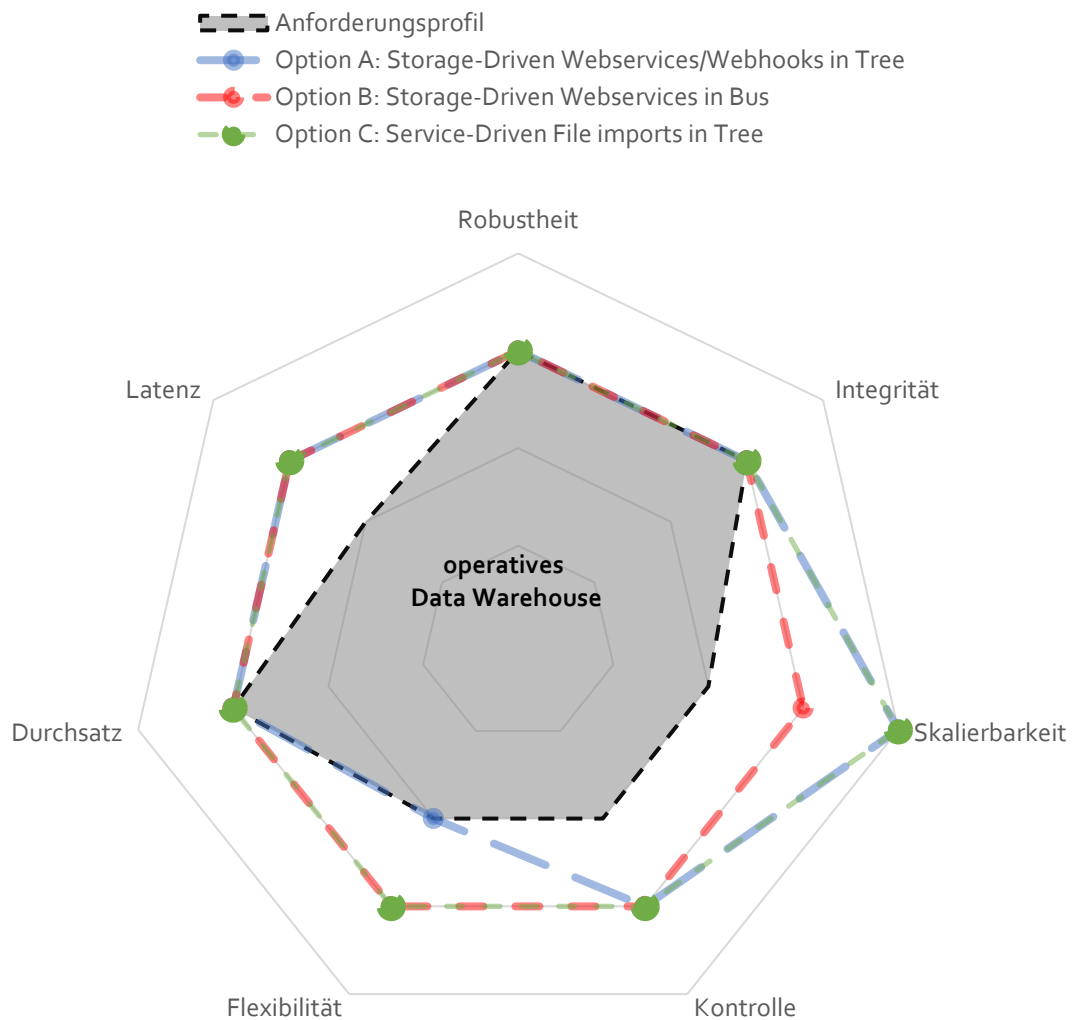


Abbildung 45: Integrationsarchitekturen für Fallstudie 1 mit verschiedenen Paradigmen und Methoden im Vergleich

6.2.3. Beschreibung der tatsächlichen Implementierung

Bei dem untersuchten System handelt es sich um ein Data Warehouse im Bereich Forschung und Entwicklung, welches Daten für systemübergreifende Anwendungen im Kontext von operativen Dashboards und Reports bereitstellt. Die Integrationsarchitektur ist in Abbildung 46 zur Veranschaulichung schemenhaft mit archetypischer Vereinfachung aufgearbeitet.

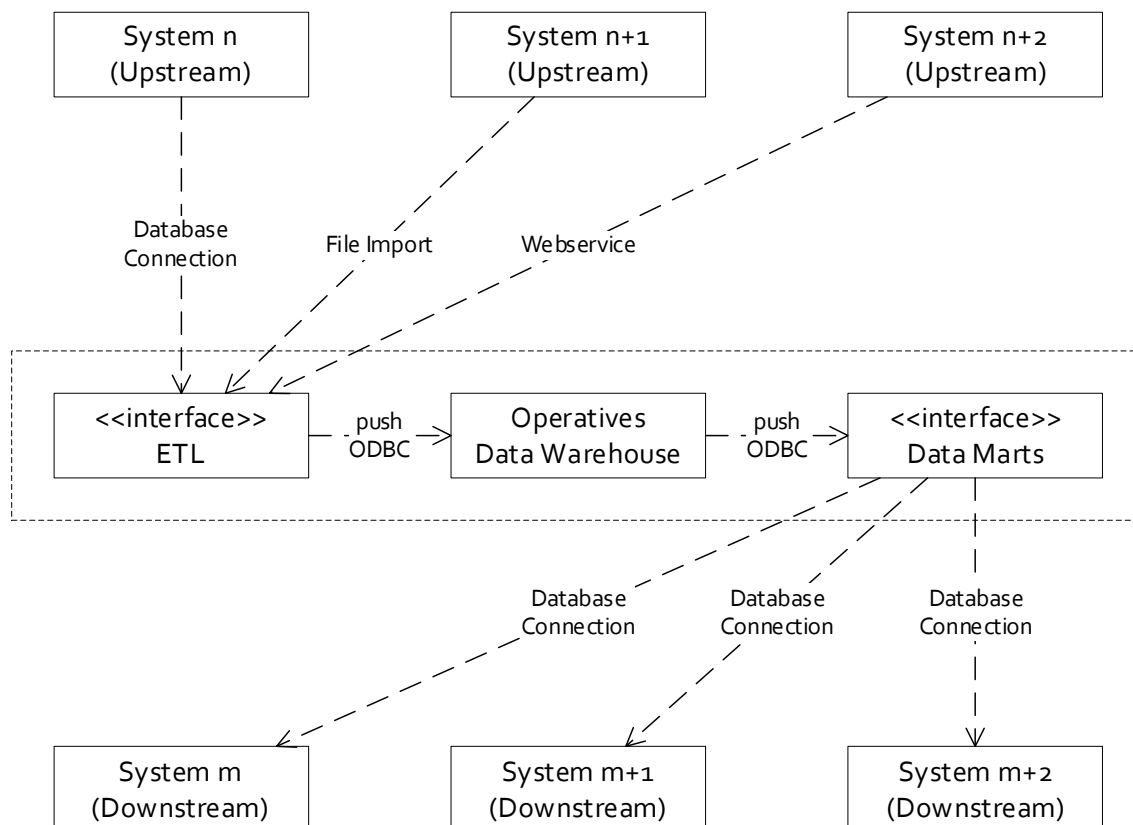


Abbildung 46: Implementierung des operativen Data Warehouses im Fallbeispiel

Im betrachteten Anwendungsfall wurde das Paradigma der Storage-Driven-Integration in einer Hub-Topologie umgesetzt.

Das operative Data Warehouse greift zyklisch aktiv auf die verbundenen Systeme zu und integriert die Daten entsprechend dem ETL-Prinzip (Extract, Transform, Load). Die Weitergabe von Informationen erfolgt dabei nur in eine Richtung. Dabei sammelt das operative Data Warehouse Daten aus den operativen Kernsystemen („Upstream“) und stellt diese für Sekundärsysteme („Downstream“) zur Verfügung, ohne dass Daten von einem angebotenen System zu einem anderen angebotenen System ausgetauscht werden.

Für die Integrationsmethode werden verschiedene Formate gewählt, wobei die Kommunikationsrichtung stets nach dem Pull-Pattern erfolgt. Die gewählte Integrationsmethode hängt im Wesentlichen von den verfügbaren Schnittstellenfähigkeiten der verbundenen Systeme ab, wobei Integrationen über Webservices und direkte Datenbankzugriffe (Database-Connection) datenbankbasierter Kommunikation (File-Import) vorgezogen wird.

Im betrachteten Anwendungsfall entspricht die Implementierung dementsprechend folgenden Architekturarchetypen:

- Storage-Driven File Import in Integration Tree
- Storage-Driven Webservices in Integration Tree
- Storage-Driven Database Connection in Integration Tree

Dabei fußen ältere bestehende Integrationen hauptsächlich auf File-Imports und Database-Connections, wobei im betrachteten Anwendungsfall aufgrund einer Cloud-first IT-Strategie bei neueren Integrationen Webservices die Vorzugsmethode sind.

6.2.4. Vergleich mit den Ergebnissen des Application Assessments

Im betrachteten Anwendungsfall wurden drei verschiedene Integrationsarchitekturen implementiert, wobei es sich bei einer um die vom GAP-Modell vorgeschlagene Lösung handelt. Abbildung 47 zeigt diese im Vergleich zu Anforderungsprofil.

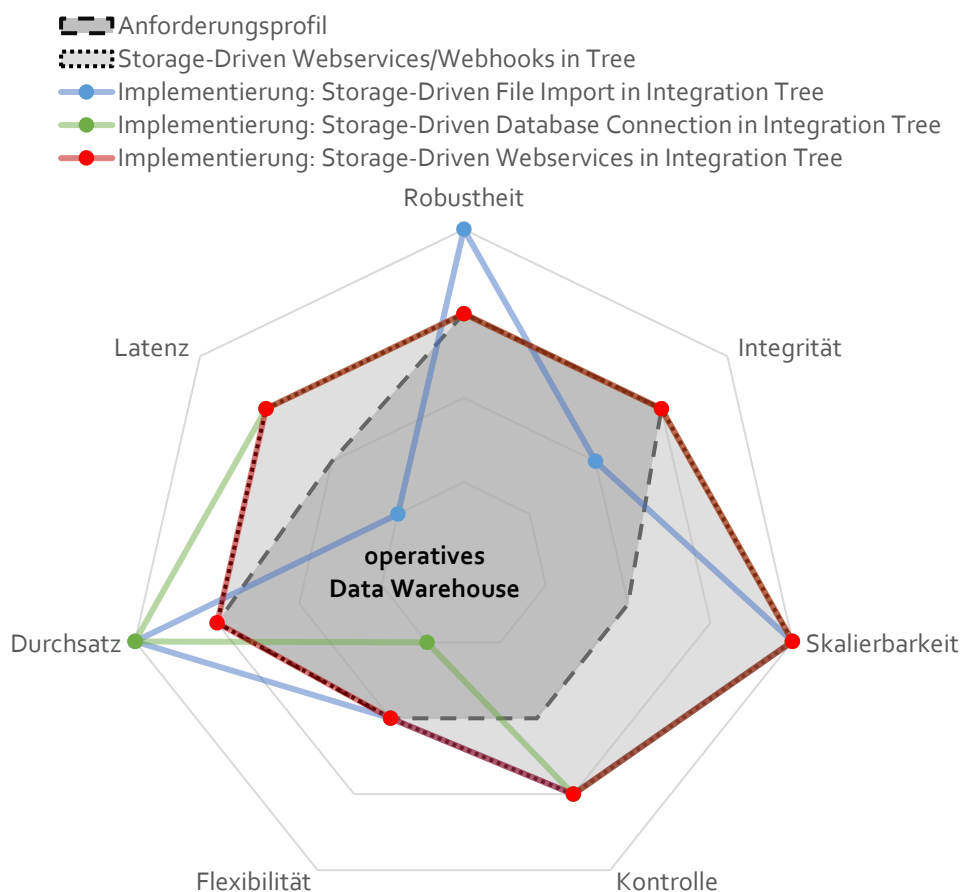


Abbildung 47: Auswertung der Ergebnisse für ein operatives Data Warehouse

Das operative Data Warehouse sammelt in der derzeitigen Implementierung in regelmäßigen Zyklen aktiv den aktuellen Datenstand direkt bei den Quellsystemen ein, wobei verschiedene Integrationsmethoden genutzt werden.

Mit Blick der Ergebnisse des GAP-Modells wäre beim Application Assessment zu erwarten, dass die gewählte Integrationsarchitektur in der aktuellen Form (Webservices und Database-

Connections) den Anforderungen entspricht, wohingegen insbesondere bei bestehenden datei-basierten Integrationen Handlungsbedarf besteht. Die Implementierung mit Storage-Driven File-Imports in einem Integration-Tree ist zwar sehr robust und ermöglicht sehr hohen Durchsatz, zeigt jedoch Schwächen bei der Latenz. Storage-Driven Database-Connections im Integration-Tree erfüllen die Anforderungen nicht in Bezug auf Flexibilität. Hier wäre zu erwarten, dass derzeitige Integrationen gut funktionieren, jedoch nicht einfach für zukünftige Anforderungsänderungen angepasst werden können.

Abbildung 48 zeigt die Ergebnisse des erfolgten Application Assessments nach TOGAF. Hierbei sind insbesondere die Einschätzungen zu „API First“, „Security and Compliance“ sowie „Tech Fit for now“ und „Tech fit for future“ relevant.

The screenshot displays the results of an Application Assessment for 'operatives Data Warehouse'. The assessment is organized into three columns of criteria, each with a status indicator (checkmark) and a coverage or fit rating.

Criteria	Off-the-shelf	API First
Application *	Major Gap in Coverage	Major Gap in Coverage
Modular Architecture	Major Gap in Coverage	
Security and Compliance	Near-complete Coverage / Acceptable	
Training and documentation	Minor Gap in Coverage	
Operational Excellence	Minor Gap in Coverage	
Replaced by		
Plan for Decommission		
URL to the application architecture		
BeatID		
Business fit for now		Low
Business fit for future		med
Hosting type		
Clinical Development Process		
Development function		

Additional details from the screenshot:

- Operatives Data Warehouse**: Major Gap in Coverage
- Modular Architecture**: Major Gap in Coverage
- Security and Compliance**: Near-complete Coverage / Acceptable
- Training and documentation**: Minor Gap in Coverage
- Operational Excellence**: Minor Gap in Coverage
- Replaced by**: Enter value here
- Plan for Decommission**: Enter value here
- URL to the application architecture**: Enter value here
- BeatID**: Enter value here
- Business fit for now**: med
- Business fit for future**: low
- Hosting type**: on-prem
- Clinical Development Process**: Design, Conduct, Analyze, Setup, Submit
- Development function**: CD&O

Abbildung 48: Ergebnisse des Application Assessments für das operative Data Warehouse
(Quelle: Eigener Screenshot aus einem internen Expertensystem)

Die bestehenden Implementierungen mit File-Imports und Database-Connection werden vom GAP-Modell in Bezug auf Latenz bzw. Flexibilität schlecht bewertet. Dieses Ergebnis deckt sich mit der Bewertung von „API-First“ als „*Major Gap in Coverage*“ (Anforderungen weit verfehlt). API-First beschreibt, wie gut die implementierten Integrationsarchitekturen für ein modernes Konnektivitätsökosystem geeignet sind. Eine Vielzahl an Integrationen verfehlt das Anforderungsprofil in den relevanten Bereichen. Insbesondere File-Imports haben sich als sehr fehleranfällig erwiesen, da sich vor allem beim Einlesen und Interpretieren der Dateien kleinste Änderungen an den Quelldaten zum Abbruch des Integrationsprozesses geführt haben. Datenbankzugriffe haben sich in der Vergangenheit als robuste Alternative erwiesen. Die fehlende Flexibilität und eingeschränkte Abstraktion führt jedoch selbst bei kleinsten Änderungen immer zu Anpassungen an der Transformationslogik, was zu hohen Wartungskosten führt. Zusätzlich gibt es bei dateibasierten Integrationen aufgrund fehlender Zeitstempel Probleme mit der Synchronität und Integrität der Daten. Für zukünftige Anbindungen werden Webservices präferiert. Die ausgetauschten Datenmengen sind im Kontext eines operativen Data Warehouses bei täglichen Delta-Load überschaubar, so dass nur eine geringe Last auf die angebundenen Systeme wirkt. Diese lässt sich zusätzlich durch geschicktes „Schedulen“ der ETL-Prozesse minimieren. Außerdem ermöglicht die Abstraktion in Webservices die Entkopplung der Integration von der technischen Funktionalität der jeweiligen operativen Kernsysteme, was die Robustheit des gesamten Systems weiter stärkt.

Das Kriterium „Security and Compliance“ beschreibt, wie gut die derzeitige Implementierung den Anforderungen an Datensicherheit und -qualität abbildet. Im Kontext des GAP-Modells kann dies auf die Kriterien Robustheit und Kontrolle übertragen werden. Die gute Bewertung der tatsächlichen Implementierung deckt sich mit den Ergebnissen des GAP-Modells.

Das Kriterienpaar „Tech fit for now“ (derzeit) und „Tech fit for future“ (zukünftig) beschreibt, wie gut die tatsächliche Implementierung den derzeitigen und zukünftigen Anforderungen gerecht wird. Dabei wird der derzeitige Technical Fit als mittelmäßig („*med*“) bewertet und fällt für die Zukunft auf niedrig („*low*“) ab. Dies bestätigt die Beobachtung, dass nur die kürzlich implementierten Integrationen den Anforderungen nach GAP-Modell entsprechen, die restlichen Integrationen die Anforderungen knapp verfehlen und daher nicht nachhaltig und zukunftssicher implementiert sind.

In diesem Anwendungsfall stimmen die Ergebnisse des Application Assessments mit den Bewertungsergebnissen des GAP-Modells überein. Das GAP-Modell liefert automatisiert passende Architekturvorschläge und identifiziert korrekt Stärken und Schwächen der bestehenden Integrationen. Es zeigt Handlungsbedarf auf und liefert in diesem Fall mit geringem Aufwand eine nützliche Unterstützung. Die Ergebnisse der ersten Fallstudie bestätigt somit die zentrale These dieser Arbeit.

6.3. Anbindung von Satellitensystem an operative Kernsysteme

In diesem betrachteten Anwendungsfall geht es um die Anbindung von Satellitensystemen an operative Kernsysteme. Abbildung 49 zeigt eine Übersicht des Vorgehens und der Ergebnisse dieser Fallstudie 2. Aus dem individuellen Anforderungsprofil ergibt sich ein Optionsraum mit 72 möglichen Architekturarchetypen, welche anhand der gewichteten Kriterien bewertet werden. Aus dieser Menge können vier geeignete Integrationsarchitekturen identifiziert werden. Die Empfehlung des GAP-Modells ist eine Ertüchtigung des Kernsystems zur serviceorientierte web-basierte Anbindung der Satelliten über einen Integration-Hub. Eine detaillierten Zwischenergebnisse bei der Durchführung der Bewertung nach GAP-Modell finden sich in Anhang 12 ab Seite 161.

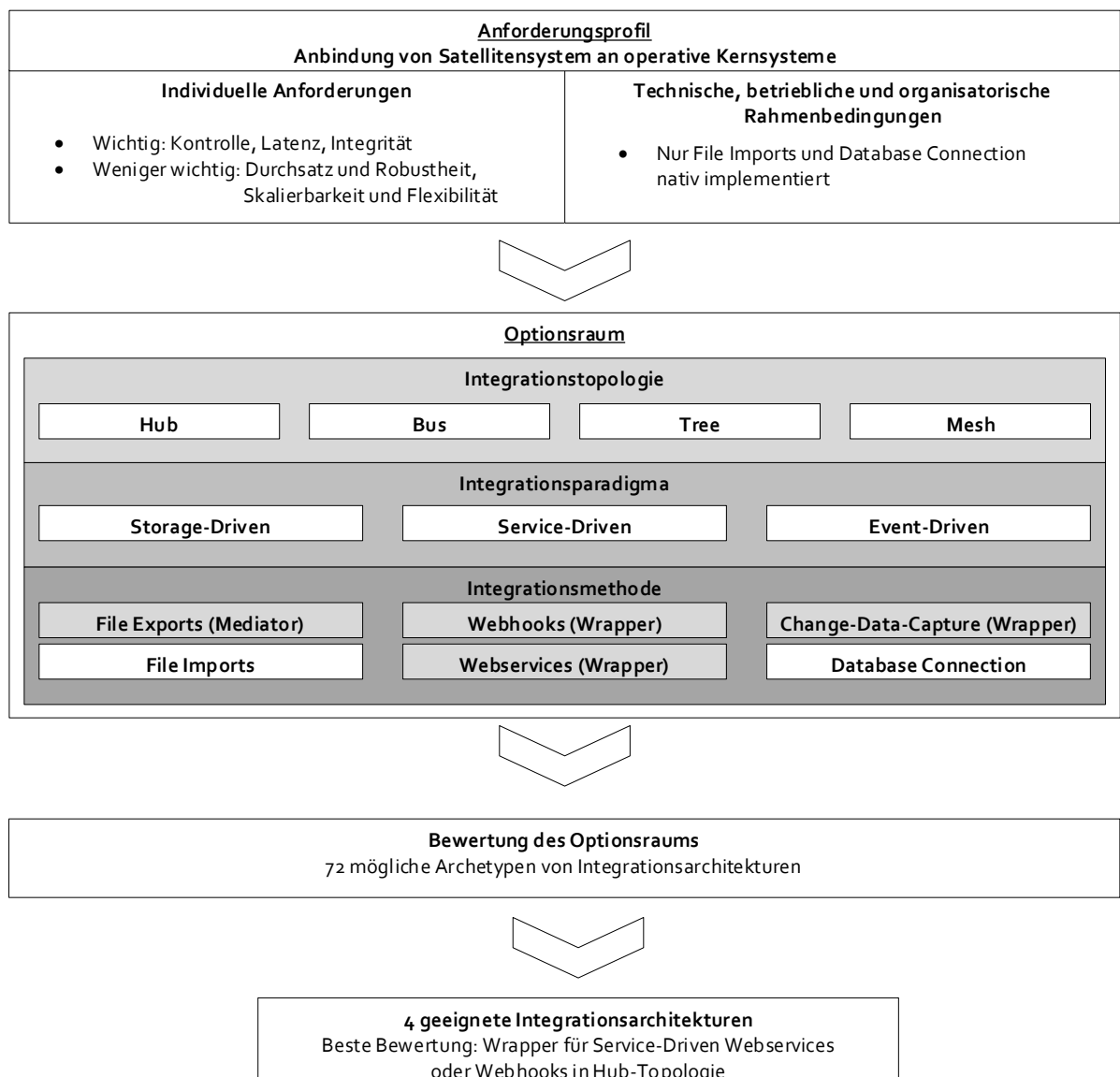


Abbildung 49: Vorgehen und Ergebnisse von Fallstudie 2

6.3.1. Beschreibung der Anforderungen

In Folgendem werden die Anforderungen, die an eine Integrationsarchitektur zur Anbindung von Satellitensystemen an ein operatives Kernsystem beschrieben. Bei diesem untersuchten Fallbeispiel handelt es sich um die Anbindung von Drittsystemen an operative Kernsystem. Die Drittsysteme, in diesem Kontext Satelliten genannt, sind als Consumer an ein oder mehrere zentrale Provider-Systeme gekoppelt, in welchen die Kerngeschäftsprozesse, -daten und -logiken abgebildet sind. Abbildung 50 zeigt das individuelle Anforderungsprofil mit gewichteten Kriterien.

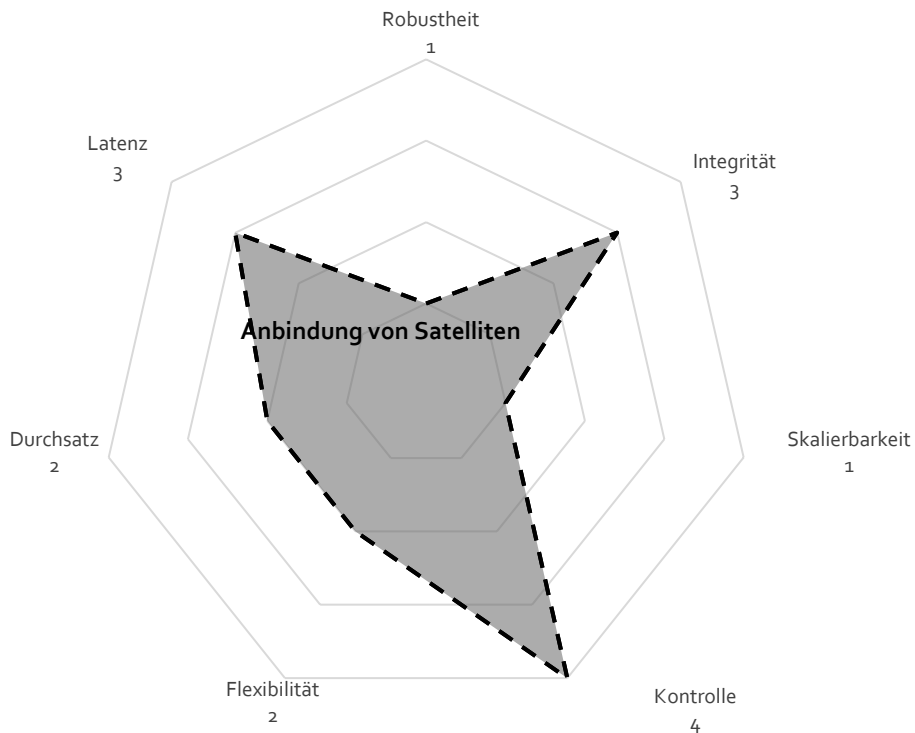


Abbildung 50: Anforderungsprofil an Integrationen zur Anbindung von Satelliten

Wichtigstes Kriterium für eine geeignete Integrationsarchitektur für den betrachteten Anwendungsfall ist die *Kontrolle* (siehe Abbildung 50). Passende Integrationsarchitekturen müssen in dieser Dimension mit der maximalen Wertzahl 4 (*sehr stark*) bewertet sein. Diese Anforderung ergibt sich, aus der Abbildung GxP-relevanter Prozessen im Kernsystem. Es handelt sich um ein validiertes Computersystem. Dies bedeutet, dass alle nötigen Maßnahmen getroffen werden müssen, um die Richtigkeit und Verlässlichkeit der Daten jederzeit zu gewährleisten. Dies beinhaltet insbesondere die Fähigkeit alle Datenveränderungen zu dokumentieren, um fälschliche Manipulationen erkennen und korrigieren zu können. [1, Seite 6–8] In Bezug auf Systemintegration folgt daraus, dass die Informationsintegration jederzeit kleinteilig überwacht, gesteuert und dokumentiert werden muss.

Zweitwichtigste Kriterien sind die Anforderungen an *Latenz* und *Integrität* der Integrationen zwischen Satelliten und operativen Kernsystem. Die Zielbewertung liegt in dieser Dimension bei der Wertzahl 3 (*stark*). Dies ergibt sich aus der engen Integration der Satelliten mit dem Kernsystem. Das operative Kernsystem verwaltet die wichtigsten Daten und stellt Kernfunktionalitäten bereit. Die implementierten Geschäftsprozesse werden durch die Satelliten erweitert und teilweise automatisiert. Die Satelliten kombinieren APIs verschiedener angebundener Systeme, führen Informationen zusammen verknüpfen Geschäftsprozessen über Systemgrenzen hinweg. Latenz und Integrität sind dabei eng miteinander verbunden. Beides ist nötig, damit Daten und Funktionalität anderer Systeme nahtlos in den im Satelliten abgebildeten Prozess integriert werden

können. Integrationen mit einer guten Latenz weisen nur einen minimalen zeitlichen Verzug auf. Integrationen mit einer guten Integrität vermeiden Datenkonflikte, Kollisionen und Systembrüche.

Für die gesuchte Integrationsarchitektur sind *Durchsatz* und *Flexibilität* weniger wichtig und können mit der Wertzahl 2 (*schwach*) bewertet sein. Ein hoher Durchsatz ist nicht nötig, da aufgrund der Anwendung für Prozessintegration keine großen Datenmengen aus dem operativen Kernsystem extrahiert werden. Auch die Dimension Flexibilität ist für die Auswahl der Integrationsarchitektur nicht ausschlaggebend, da sich das Kernsystem in seiner Struktur und Funktionalität wenig ändert.

Aspekte der *Robustheit* und *Skalierbarkeit* spielen bei der Auswahl der Integrationsarchitektur keine Rolle und können mit der Wertzahl 1 (*sehr schwach*) bewertet sein. Alle wichtigen Daten werden im operativen Kernsystem verwaltet und so sind robuste Integrationen nicht entscheidend. Auch die Skalierbarkeit ist kein ausschlaggebendes Kriterium, da das Prozessaufkommen planbar ist und schnelle Veränderungen nicht zu erwarten sind.

Aus technischer Sicht verfügt betrachtete operative Kernsystem nicht über eine nativ implementierte Web-API. Stattdessen ist der Zugriff auf Daten des Kernsystems über Datenbankzugriff oder über vom System bereitgestellte Dateieexporte vorgesehen. Andere Integrationsmethoden, außer File-Imports und Database-Connections sind daher nur mithilfe eines API-Wrappers realisierbar.

6.3.2. Anwendung des GAP-Modells

Der in Kapitel 3 definierten Klassifizierung von Designartefakten folgend wurden durch das GAP-Modell automatisch alle möglichen Architekturausprägungen generiert und miteinander bewertet. Eine detaillierte Bewertung des gesamten Optionsraums finden sich in Anhang 12 ab Seite 161.

Das GAP-Modell identifiziert sechs Archetypen als potenziell geeignete Kandidaten für die Integrationsarchitektur zur Anbindung von Satellitensystemen an ein operatives Kernsystem. Die sehr hohe Anforderung an Kontrolle erfüllen alle Optionen. Vier von sechs Optionen erreichen die Maximalbewertung auch bei Integrität und Durchsatz. Die niedrigen Anforderungen an Flexibilität und Skalierbarkeit werden bei allen Optionen übererfüllt.

Tabelle 14 zeigt die Bewertungsergebnisse für passenden Integrationsarchitekturen. Es werden pro Zeile die Wertpunkte der entsprechenden archetypischen Integrationsarchitektur für jedes Kriterium im Vergleich zum Anforderungsprofil gezeigt und mit einer allgemeinen Bewertung ergänzt.

				Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Anforderungsprofil				1	3	1	4	2	2	3	
Architekturarchetypen	Hub	Service-Driven	Webhooks	1	4	2	4	4	2	4	8,36
	Hub	Service-Driven	Webser-vices	1	4	2	4	4	2	4	8,36
	Hub	Event-Driven	Webhooks	1	4	2	4	4	2	4	8,36
	Hub	Event-Driven	Webser-vices	1	4	2	4	4	2	4	8,36
	Hub	Storage-Driven	Webhooks	2	3	2	4	3	3	3	7,87
	Hub	Storage-Driven	Webser-vices	2	3	2	4	3	3	3	7,87

Tabelle 14: Bewertungsergebnisse des GAP-Modells für Integrationen zur Anbindung von Satelliten

Die Ergebnisse des GAP-Modells zeigen eine klare Favorisierung der Hub-Topologie und webbasierter Integrationsmethode. Unterschiede in der Bewertung ergeben sich aus der gewählten Integrationstopologie, wobei Event-Driven und Service-Driven zu denselben Werten führen und insgesamt etwas besser bewertet sind als Storage-Driven-Integrations. Idealerweise wird also ein API-Wrapper für das operative System implementiert, welcher die Daten und Funktionalitäten für Satelliten als Services zur Verfügung stellt und gleichzeitig Änderungen an eine zentrale Instanz kommuniziert. Alle Verbindungen sollten dabei über eine zentrale Komponente erfolgen, so dass der Zugriff zentral erfasst, verwaltet und überwacht werden kann.

Abbildung 51 zeigt die verschiedenen Profile im Vergleich zum Anforderungsprofil. Die klare Empfehlung zur Hub-Topologie ergibt sich aus der Anforderung nach sehr starker Kontrolle. Wenn die Kommunikation aller Akteure im Konnektivitätsökosystem exklusiv über einen Zentralen Integration-Hub realisiert werden, kann ein maximaler Grad an Kontrolle umgesetzt werden. Ein Integration-Hub muss somit den Empfehlungen des GAP-Modells folgend zwingend ein Teil der Integrationsarchitektur sein, um die gegebenen Anforderungen zu erfüllen.

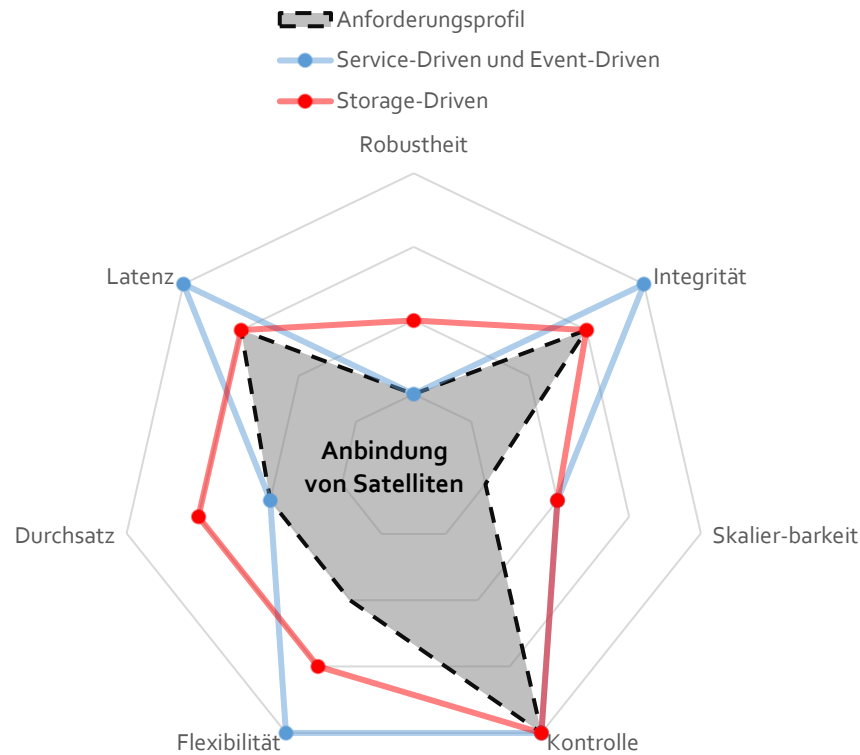


Abbildung 51: Integrationsarchitekturen für Fallstudie 2 mit verschiedenen Paradigmen im Vergleich

Für das Erreichen der Anforderungen eignen sich alle Integrationsparadigmen, wobei sich eine leichte Tendenz hin zu Event-Driven-Integrations und Service-Driven-Integrations abzeichnet. Storage-Driven-Integrations sind durch das physische Kopieren der Daten deutlich robuster als Service-Driven-Integrations. Gleichzeitig birgt das Duplizieren von Daten die Gefahr von Datenkonflikten und geht damit zu Lasten der Integrität. Da im betrachteten Anwendungsfall Integrität als beinahe doppelt so wichtig wie Robustheit gewertet wurde, ergibt sich die Favorisierung von des Service-Driven-Paradigmas.

Für die Integrationsmethode schlägt das GAP-Modell klar ein webbasiertes Format vor, wobei die Richtung der Kommunikationsinitiierung keine Rolle zu spielen scheint. Webservices und Webhooks sind gleich gut bewertet und bilden die favorisierte Integrationsmethode. Dies ist insofern interessant, als dass das operative Kernsystem über keinerlei native Unterstützung dieser Integrationsmethoden verfügt. Zur Realisierung der vorgeschlagenen Integrationsarchitektur muss das operative Kernsysteme mit einem API-Wrapper für die webbasierte Kommunikation ertüchtigt werden. Ein Wrapper geht als zusätzliches Element in der Integrationskette zwar zu Lasten der Robustheit, diese ist jedoch im betrachteten Anwendungsfall nicht ausschlaggebend. Dem entgegen stehen die nativ verfügbaren Integrationsmethoden Database-Connection und File-Imports, welche aufgrund der schwachen Bewertung in der Dimension Kontrolle vom GAP-Modell nicht empfohlen werden.

6.3.3. Beschreibung der tatsächlichen Implementierung

Das betrachtete operative Kernsystem ist ein Clinical Trial Management System und bildet damit das Rückgrat für alle Aktivitäten, die im Zusammenhang mit Klinischen Studien stehen. Eine anonymisierte Darstellung der betrachteten Integrationslandschaft findet sich in Anhang 12 auf Seite 161.

Die tatsächliche Implementierung des betrachteten Anwendungsfall ist historisch gewachsen und mit 42 Systemen verbunden. 28 dieser Systeme sind Consumer bzw. Prosumer des operativen Kernsystems. Unter Anwendung der Taxonomie des GAP-Modells lassen sich diese 28 ausgehenden Integrationen zu vier verschiedenen Archetypen zuordnen. Abbildung 52 zeigt diese in vereinfachter Weise. In Rahmen dieser Fallstudie wird die Verbindung zu vier ausgewählte Satellitensysteme betrachtet, welche exemplarisch für die entsprechenden Archetypen stehen.

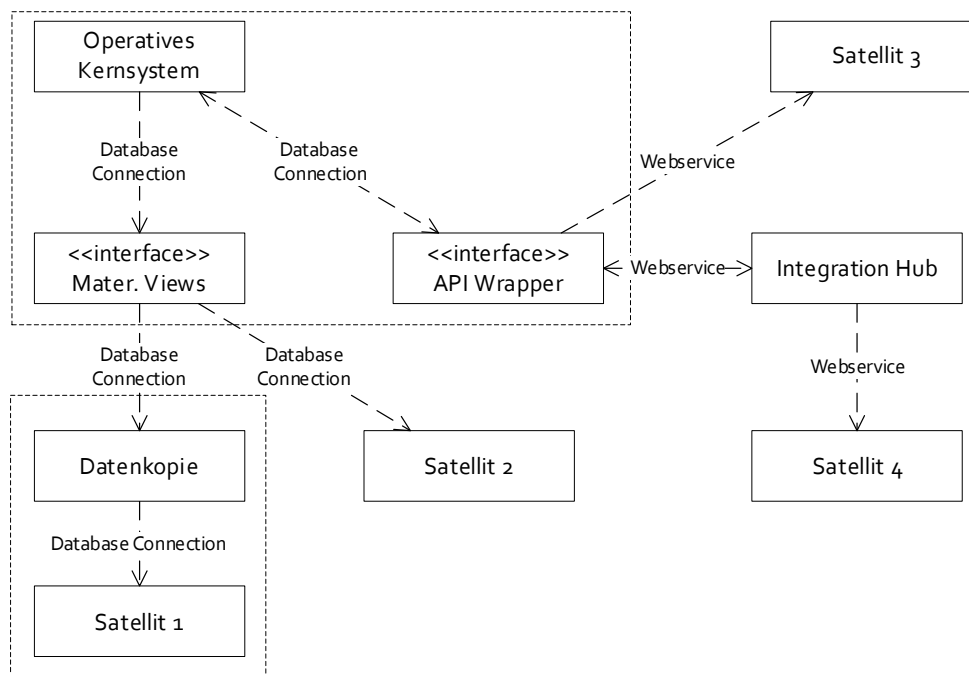


Abbildung 52: Implementierung der Integration von Satellitensystemen

Aus technischer Sicht verfügt das Kernsystem nur über eingeschränkte native Schnittstellen. Neben der Möglichkeit des Zugriffs auf bereitgestellte Datenexporte im CSV-Dateiformat gibt es nur die Möglichkeit lesender Datenbankzugriffe Materialized Views der Produktivdatenbank, sowie die Ausführung von vorbereiteten PL/SQL-Funktionen¹ auf Datenbankebene. Materialized Views sind in diesem Kontext exakte Kopien der Tabellen und Sichten der Produktivdatenbank, welche periodisch oder durch Transaktionen ausgelöst aktualisiert werden. In diesem Anwendungsfall ist eine Synchronisierung alle 15 Minuten implementiert. Somit sind ohne Wrapper lediglich die Integrationsmethoden File-Imports und Database-Connection möglich.

¹ PL/SQL ist eine prozedurale Sprache, die programmierbare Interaktionen innerhalb einer Oracle Datenbank ermöglicht. [81].

Satellit 1

Satellit 1 steht exemplarisch für 15 Integrationen, die als Storage-Driven Database-Connection in einem Integration-Tree implementiert sind.

Beim betrachteten Beispiel handelt es sich um ein System zum Management von Sequenzierungsinformationen von biologischen Proben. Diese Applikation benötigt zur Einordnung in den Kontext der jeweiligen klinischen Studie lesenden Zugriff auf die Stamm- und Metadaten im Kernsystem. Die benötigten Daten werden von diesem System nach pull-Pattern über ODBC direkt vom Kernsystem angefragt und in die Applikationsdatenbank des Satelliten kopiert. Die Daten des Kernsystems werden dabei täglich synchronisiert und vom Satelliten nicht an andere Systeme weitergegeben. Diese und ähnliche Integration funktioniert sehr robust und stellen die benötigten Daten zuverlässig zur Verfügung. Nachteilig ist, dass Datenänderungen immer über eine Nutzerinteraktion im Kernsystem erfolgen müssen, so dass es zu Systembrüchen im Prozessablauf kommt. Soll beispielsweise der Status eines Meilensteins aktualisiert werden, erfordert dies immer den Login ins Kernsystem. Ein direkter schreibender Datenbankzugriff ist aus Sicherheitsgründen nicht möglich, da die Kontrollmechanismen hierfür nicht ausreichend sind. Zusätzlich muss die Implementierung immer wieder an kleine Änderungen in der Datenstruktur angepasst werden.

Satellit 2

Satellit 2 steht exemplarisch für acht Integrationen, die als Service-Driven Database-Connection in einem Integration-Tree implementiert sind.

Das betrachtete Beispiel ist ein Dashboard zur länderspezifischen Überwachung der Rekrutierungsleistungen in klinischen Studien. Die für das Dashboard benötigten Daten werden beim Öffnen des Dashboards direkt über eine ODBC-Schnittstelle abgerufen und dann vom Satellitensystem verarbeitet und dargestellt. Eine Persistierung oder Weitergabe der Daten erfolgt nicht. Die Integration funktioniert gut, ermangelt jedoch einer Schemaabstraktion. Änderungen am Datenbankschema müssen somit auch in der Konfiguration des verbundenen Systems nachgepflegt werden. Dies führt zu Einschränkungen bei Generationen-, Technologie- oder Systemwechsel des Kernsystems. Zudem ist die Kontrolle des Datenbankzugriffs eingeschränkt, da Weitergabe der Login-Informationen Unbefugten potenziell Zugriff auf relevante Daten erlaubt.

Satellit 3

Satellit 3 steht exemplarisch für zwei Integrationen, die als Service-Driven Webservice in einem Integration-Tree implementiert sind.

Das betrachtete System dient dem Management von Studien- und Projektspezifischen Zielen. In diesem Fall werden die Daten zum einen zur Prozesssteuerung (bspw. Individuelle Verantwortlichkeiten und Rollen, Zeitlinien, operativer Status) benötigt. Zum anderen werden ähnlich wie in Satellit 1 die im Satelliten 3 abgelegten Daten zu den operativen Zielen anhand der Stamm- und Metadaten aus dem Kernsystem kontextualisiert. Aufgrund der vermehrten Nachfrage nach ähnlichen Anwendungsfällen und den Einschränkungen von auf Datenbankzugriffen basierenden Integrationen wurde ein API-Wrapper implementiert. Diese Middleware interpretiert die Anfragen an den REST-basierten Webservice und überträgt diese in SQL-Anfragen. Die Middleware ist separat vom Kernsystem implementiert, liegt jedoch im selben betrieblichen Verantwortungsbereich und ist somit Teil des Systembereichs. Im Satellitensystem selbst werden nur die IDs der entsprechenden Objekte des Kernsystems persistiert. Werden die Objektdaten benötigt, dann fragt das Satellitensystem diese direkt beim Kernsystem an. Dabei lässt sich spezifizieren, welche Attribute benötigt werden, so dass die Anfragen dahingehend optimiert werden können. Auch ist die Abfrage einzelner Objekte, oder aller Objekte mit Filtermöglichkeiten implementiert.

Integrationen über den API-Wrapper funktionieren sehr gut, wobei es anfänglich zu großen Problemen mit der Verfügbarkeit der Wrapper-Applikation gab. Erst durch Anpassungen an Technologie und Betriebsumfeld konnte eine Stabilisierung des Betriebs erreicht werden, welche jedoch nicht an die Robustheit der einfachen Datenbankzugriffe heranreicht. Auch ist mit dieser Lösung noch kein schreibender Zugriff auf das Kernsystem möglich, da ähnlich wie bei Satellit 1 die Kontrollmechanismen nicht ausreichen.

Satellit 4

Satellit 4 steht exemplarisch für vier Integrationen, die als Service-Driven Webservice in einem Integration-Hub implementiert sind.

Bei der betrachteten Applikation handelt es sich um ein Produkt zur Workflow-Automatisierung, welches die Prozesse im Kernsystem erweitert, vereinfacht, mit Informationen aus anderen Quellen anreichert und teilweise automatisiert. Zusätzlich werden Systemübergreifende Prozesse in einer gemeinsamen Applikation umgesetzt. Voraussetzung hierfür ist neben dem Echtzeitzugriff auf Daten und Funktionalität der integrierten vor allem die Möglichkeit auch schreibende Aktivitäten über die Integration auszuführen.

Die Integration funktioniert einwandfrei und nutzt den in Satellit 3 beschriebene Wrapper, welcher umschreibende Funktionalität ergänzt wurde. Zusätzlich dazu wird zyklisch vom Satelliten 4 nach bestimmten Änderungen im Kernsystem angefragt, welche dann Prozesse im Satellitensystem auslösen. Anfangs wurde hierfür die Filterfunktionalität der REST-API genutzt. Im Rahmen der kontinuierlichen Weiterentwicklung wurde dann ein hierauf optimierter Endpunkt implementiert. Der Zugriff auf den Webservice des Kernsystems erfolgt bei Satellit 4 nicht direkt, sondern über einen Integration-Hub, bzw. eine API-Management Plattform. Hier sind zentrale Mechanismen zur Authentifizierung, Überwachung und Dokumentation der Kommunikation implementiert, so dass den Anforderungen an validierte Computersysteme Rechnung getragen wird und auch Cloud-Anwendungen sicher angebunden werden können. Derzeit nutzen vier Satellitensysteme diesen Integrationspfad.

6.3.4. Vergleich mit den Ergebnissen des Application Assessments

Abbildung 53 zeigt die das Anforderungsprofil nach GAP-Modell, sowie die Bewertungsergebnisse der verschiedenen Integrationen. Dabei werden die vier ausgewählten Implementierungen zur Anbindung der Satelliten an operative Kernsysteme mit dem Anforderungsprofil und der vorgeschlagenen Integrationsarchitektur verglichen.

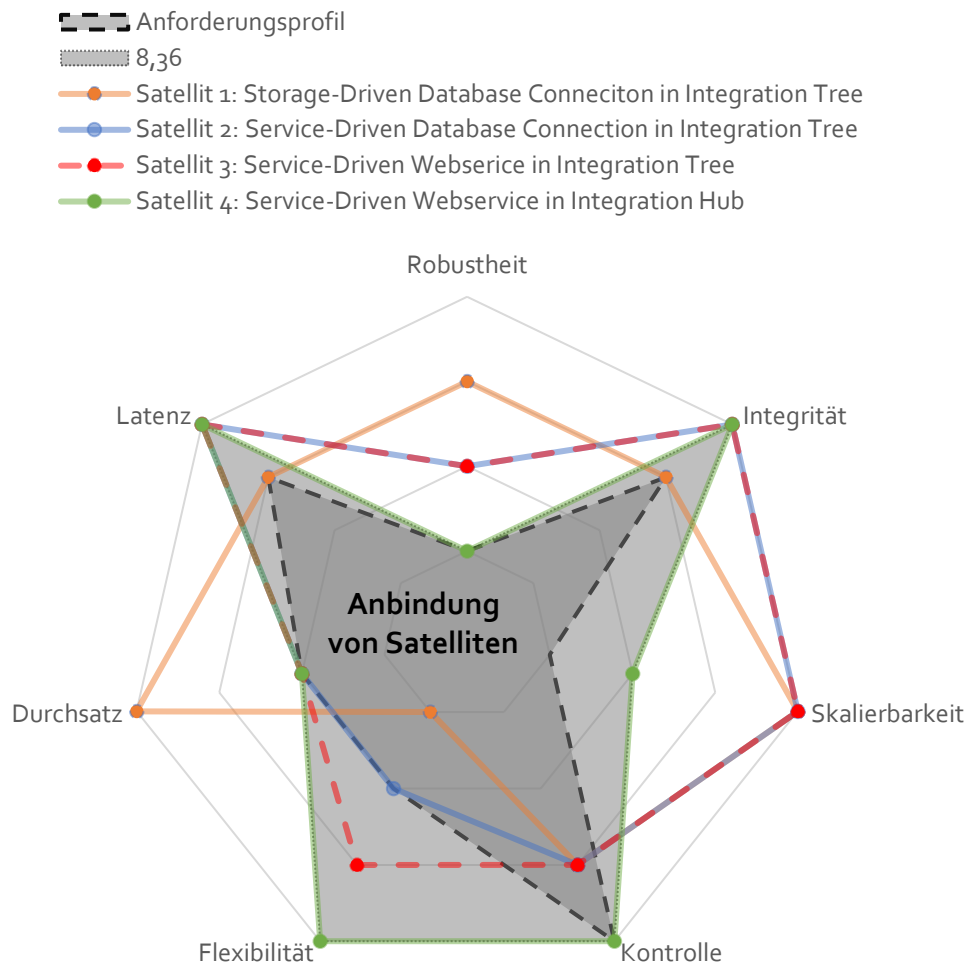


Abbildung 53: Ergebnisse für Integrationen zur Anbindung von Satelliten

Bei der derzeitigen Implementierung des Anwendungsfalls sind verschiedenen Architekturen implementiert. Bei Satellit 4 ist mit der webservice-basierten Implementierung nach Service-Driven-Paradigma in mit einem Integration-Hub die vom GAP-Modell vorgeschlagene Architektur implementiert. Dies entspricht der Empfehlung des GAP-Modells. Die anderen Implementierungen entsprechen nahezu dem Anforderungsprofil, wobei alle einen Wertpunkt unter der Anforderung an eine sehr starke Kontrolle liegen und Satellit 2 zusätzlich die relativ geringe Anforderung an Flexibilität verfehlt. In folgendem werden diese Bewertungsergebnisse in den Kontext des erfolgten Application Assessments gestellt. Dabei wird zusätzlich zum Application Assessment des operativen Kernsystems das Application Assessment der Satellitensysteme betrachtet, da in diesem Anwendungsfall die aktive Komponente der Integrationen jeweils auf Seite der Satelliten implementiert ist.

Bewertung des operative Kernsystems

Abbildung 54 zeigt die Bewertungsergebnisse für des operative Kernsystems. Dabei werden in allen für die Integrationsarchitektur relevanten Kategorien schlechte Bewertungsergebnisse erzielt. Diese Ergebnisse des Application Assessments bestätigt die in Abbildung 53 gezeigten Bewertungsergebnisse des GAP-Modells.

☒ Application * operatives Kernsystem	☒ Off-the-shelf Near-complete Coverage / Acceptable	☒ API First Major Gap in Coverage
☒ Modular Architecture Major Gap in Coverage	☒ Security and Compliance Near-complete Coverage / Acceptable	☐ Data assets/products Studies, Study Countries, Study Sites
☒ Training and documentation —	☒ Operational Excellence —	☒ Decommissioned date 2024
☒ Replaced by Veeva Gemstone	☒ Plan for Decommission ✓	☒ Tech Fit for future Low
☒ URL to the application architecture Enter value here	☒ BeatID BEAT00322776	☒ Tech Fit for now low
☒ Business fit for now —	☒ Business fit for future —	☒ data follow pattern —
☒ Hosting type on-prem	☒ Clinical Development Process Design Setup Conduct Analyze	☒ Development function —

Abbildung 54: Application Assessment für das operative Kernsystem
 (Quelle: Eigener Screenshot aus einem internen Expertensystem)

Die dokumentierte Entscheidung zur Dekommissionierung zeigt, dass die bestehende Implementierung das Anforderungsprofil nicht mehr erfüllt. Deckungsgleich verfehlen im GAP-Modell nahezu alle Implementierungen die Anforderungen. Lediglich serviceorientierte Webservices über einen Integration-Hub erfüllen alle Kriterien des GAP-Modells, wobei hier nur eine sehr schlechte Robustheit erreicht werden kann (Beispiel: Satellit 4).

Unter den gegebenen technischen, betrieblichen und organisatorischen Rahmenbedingungen sind passende Integrationsarchitekturen also nur schwer realisierbar. Entsprechend werden im Application Assessment „API-First“ und „Modular-Architecture“ als „Major Gap in Coverage“ (Anforderungen weit verfehlt) bewertet. Da eine native Web-API nicht verfügbar ist, kann die Kapselung der Daten und Funktionalität nur mithilfe eines Wrappers realisiert werden. Das Kernsystem erfüllt zwar die Sicherheitsanforderungen, jedoch nur in der derzeitigen Laufzeitumgebung. Bestehenden Integrationen können nur „On-Premise“, oder in einer privaten Cloud sicher und compliant betrieben werden, was im Kontext der Verlagerung in die Cloud problematisch ist. Eine Ertüchtigung mithilfe einer Middleware zur API-fizierung ist wie oben beschrieben möglich, jedoch nur eine Zwischenlösung. Dabei wird die operative Integrität aufrechterhalten, während das operative Kernsystem selbst hinter dieser Fassade ausgetauscht wird.

Satellit 1

Die Integration mit dem Satellitensystem 1 ist in Tree-Topologie nach Storage-Driven-Paradigma mit Database-Connection-Methode implementiert. Für Satellit 1 zeigt das GAP-Modell Schwächen bei der Flexibilität, sowie der Kontrolle. Die in Abbildung 55 dargestellten Ergebnisse des Application Assessments bestätigen diese Einschätzung.

☐ Application * Satellit 1	☒ Off-the-shelf Major Gap in Coverage	☒ API First Major Gap in Coverage
☒ Modular Architecture Major Gap in Coverage	☒ Security and Compliance —	☐ Data assets/products Enter value here
☒ Training and documentation —	☒ Operational Excellence —	☐ Decommissioned date Enter value here
☐ Replaced by Enter value here	☐ Plan for Decommission	☒ Tech Fit for future Low
☐ URL to the application architecture Enter value here	☐ BeatID BEAT03001944	☒ Tech Fit for now med
☒ Business fit for now —	☒ Business fit for future —	☒ data follow pattern Data Prosumer
☒ Hosting type on-prem	☐ Clinical Development Process	☒ Development function —

Abbildung 55: Ergebnisse des Application Assessment für Satellit 1
(Quelle: Eigener Screenshot aus einem internen Expertensystem)

Die derzeitige Erfüllung der Anforderungen („Tech Fit for now“) ist als mittel (*med*), bewertet, wobei dieser in Zukunft („Tech Fit for future“) auf niedrig (*low*) fallen wird. Diese Einschätzung bestätigt, dass das Anforderungsprofil nur knapp verfehlt wird, die Integration bei Änderungen am operativen Kernsystem jedoch nicht mehr geeignet sein wird. Zwar ist die Integration robust implementiert, es fehlt jedoch an Kontrolle und Flexibilität. Ausschlaggebend hierfür sind sowohl die datenbankbasierte Integrationsmethode, als auch das Storage-Driven Paradigma.

Die fehlende Möglichkeit zur webbasierten Anbindung schlägt sich in der sehr schlechten Bewertung des Punktes „API First“ mit „Major Gap in Coverage“ (Anforderungen weit verfehlt) nieder. Die Ergebnisse des Application Assessment passen somit gut zur Bewertung des GAP-Modells, wobei in der derzeitigen Version des GAP-Modell die Hosting-Umgebung keine Rolle spielt, obwohl diese einen Einfluss auf die Eignung der Integrationsmethode zu haben scheint. Wie oben beschrieben, steht die gewählte Implementierung jedoch einer zukünftigen Migration in die Cloud im Wege.

Satellit 2

Die Integration mit Satellit 2 wurde als Service-Driven Database-Connection in einem Integration-Tree implementiert. Das GAP-Modell bewertet in der Kategorien Kontrolle als nicht ausreichend, wobei alle anderen Anforderungen erfüllt werden. In der Praxis funktioniert die Integration stabil, ist jedoch nur eingeschränkt kontrollierbar und erfordert bei Änderungen der Datenstruktur Anpassungen an der Konfiguration. Die in Abbildung 56 gezeigten Ergebnisse des Application Assessments bestätigen größtenteils dieses Bild.

☒ Application * Satellit 2	☒ Off-the-shelf Major Gap in Coverage	☒ API First Major Gap in Coverage
☒ Modular Architecture Near-complete Coverage / Acceptable	☒ Security and Compliance Complete Coverage / Satisfactory	☐ Data assets/products Patient Recruitment / Operational Performance
☒ Training and documentation Complete Coverage / Satisfactory	☒ Operational Excellence Complete Coverage / Satisfactory	☐ Decommissioned date Enter value here
☐ Replaced by Enter value here	☐ Plan for Decommission	☒ Tech Fit for future Med
☐ URL to the application architecture Enter value here	☐ BeatID BEAT03001742	☒ Tech Fit for now med
☒ Business fit for now high	☒ Business fit for future high	☒ data follow pattern Data consumer
☒ Hosting type on-prem	☐ Clinical Development Process Conduct	☒ Development function CD&O

Abbildung 56: Ergebnisse des Application Assessment für Satellit 2
 (Quelle: Eigener Screenshot aus einem internen Expertensystem)

Die Applikation ist in Bezug auf „Coverage bei API-First“ schlecht bewertet („Major Gap in Coverage“ - Anforderungen weit verfehlt). Dies korrespondiert mit der Bewertung von Kontrolle als nicht ausreichend im GAP-Modell.

Dem gegenüber steht, dass die Applikation die Anforderungen an „Security and Compliance“ voll erfüllt, was jedoch insbesondere durch das on-Premise-Hosting erreicht wird. Sowohl operatives Kernsystem als auch angebundenes Satellitensystem werden in einem gemeinsamen gesicherten Bereich betrieben werden. Diese Bewertung lässt somit eine zentrale Anforderung im GAP-Modell außer Acht, bestätigt aber die gute Bewertung in Bezug auf Robustheit und Integrität im GAP-Modell.

Im GAP-Modell ist die Flexibilität als gerade ausreichend bewertet. Auch diese Bewertung stimmt mit den Ergebnissen des Application Assessment überein. Es attestiert bei Modular Architecture ein akzeptables Ergebnis.

Satellit 3

Das GAP-Modell bewertet die Integrationsarchitektur von Satellit 3 in allen Punkten bis auf Kontrolle als ausreichend, wobei beim Kriterium Flexibilität eine höhere Bewertung als bei Satellit 2 erreicht wird. Implementiert wurde eine Integration mit Service-Driven Webservices in Tree-Topologie und API-Wrapper. Die tatsächlich implementierten Integrationen sind zweckmäßig. Die größere Flexibilität der webbasierten Integrationsmethode steht dabei einer geringeren Robustheit aufgrund des eingesetzten API-Wrappers entgegen. Die Bewertung des GAP-Modells deckt sich mit den in Abbildung 57 gezeigten Ergebnissen des Application Assessments.

☒ Application * Satellit 3	☒ Off-the-shelf Minor Gap in Coverage	☒ API First Complete Coverage / Satisfactory
☒ Modular Architecture Complete Coverage / Satisfactory	☒ Security and Compliance Minor Gap in Coverage	☒ Data assets/products performance goals of study teams and clinical teams
☒ Training and documentation Complete Coverage / Satisfactory	☒ Operational Excellence Major Gap in Coverage	☒ Decommissioned date 31.12.2022
☒ Replaced by GoalSetter 2.0 (Salesforce)	☒ Plan for Decommission ✓	☒ Tech Fit for future Low
☒ URL to the application architecture http://s1	☒ BeatID Enter value here	☒ Tech Fit for now high
☒ Business fit for now high	☒ Business fit for future high	☒ data follow pattern Data Prosumer
☒ Hosting type Private Cloud	☒ Clinical Development Process Setup Conduct Analyze Submit Design	☒ Development function CD&O

Abbildung 57: Ergebnisse des Application Assessment für Satellit 3
 (Quelle: Eigener Screenshot aus einem internen Expertensystem)

In allen relevanten Kategorien wird eine tendenziell gute Bewertung erreicht. Lediglich bei der „Operation Excellence“ liegt eine schlechte Bewertung vor, was auf Schwächen bei der Robustheit, oder Kontrolle hindeuten könnte.

Beim Hosting-Type ist noch keine Integration mit Public Cloud und SaaS möglich, da die Anforderungen an die Kontrolle nicht erfüllt werden könnten. Durch den Einsatz einer zentralen Kontrollkomponente, bspw. in Form eines API-Gateways oder Integration-Hubs ließen sich die Anforderungen an Sicherheit und Compliance auch in anderen Laufzeitumgebungen erfüllen (siehe Satellit 4). Hierbei ist jedoch die schlechte Bewertung in Bezug auf „Operational Excellence“ zu beachten. Diese zeigt, dass es im Betrieb der Applikation zu Problemen kommt. Dies korrespondiert mit der schwachen Bewertung für Robustheit im GAP-Modell.

Satellit 4

Die Implementierung der Integrationen erfolgt bei Satellit 4 mit Service-Driven Webservices in Hub-Topologie. Diese Integrationsarchitektur wird vom GAP-Modell als passend empfohlen. Die tatsächlich Implementierten Integrationen funktionieren sicher, nahezu in Echtzeit und ermöglichen eine flexible Weiterentwicklung im Kontext einer Modularen Architektur. Nachteil ist dabei die Abhängigkeit von zwei Komponenten außerhalb des Kernsystems: API-Wrapper und Integration-Hub, bzw. API-Gateway.

Abbildung 58 zeigt ein sehr gutes Bewertungsergebnis, welches sich mit den Ergebnissen des GAP-Modells deckt. Alle Integrationsrelevanten Aspekte erhalten die bestmögliche Bewertung. Die etwas schwächere Bewertung in Bezug auf „Operational Excellence“ korrespondiert mit der schwachen Bewertung für Robustheit im GAP-Modell. Die doppelte Abhängigkeit vom API-Wrapper und vom Integration-Hub erfordert erhöhten Aufwand im Betrieb. Zusätzlich sorgt diese für die geringste Robustheit aller betrachteten Integrationsarchitekturen. Auch die schlechte Bewertung im Aspekt „Off-the-shelf“ kann auf den API-Wrapper bezogen werden, korrespondiert jedoch auch mit der beschränkten Skalierbarkeit aufgrund fehlender Standardisierung der Lösung. Hierbei ist jedoch zu beachten, dass die Kategorie „Off-the-shelf“ nur mittelbar relevant für die Integrationsarchitektur ist und beschreibt, inwieweit für die Applikation Standardprodukte genutzt werden.

Application * Satellit 4	Off-the-shelf Minor Gap in Coverage	API First Complete Coverage / Satisfactory
Modular Architecture Complete Coverage / Satisfactory	Security and Compliance Complete Coverage / Satisfactory	Data assets/products Study-Specific QSD Lists
Training and documentation Near-complete Coverage / Acceptable	Operational Excellence Near-complete Coverage / Acceptable	Decommissioned date Enter value here
Replaced by QVadis	Plan for Decommission ✓	Tech Fit for future Med
URL to the application architecture https://	BeatID Enter value here	Tech Fit for now high
Business fit for now high	Business fit for future high	data follow pattern Data Prosumer
Hosting type SaaS	Clinical Development Process Conduct	Development function CD&O

Abbildung 58: Ergebnisse des Application Assessment für Satellit 4
(Quelle: Eigener Screenshot aus einem internen Expertensystem)

6.4. Kontinuierliche Synchronisierung von zweier operativer Kernsysteme

Bei dem in Folgendem betrachteten Anwendungsfall handelt es sich um die Synchronisierung zweier operativer Kernsysteme. Die Datendomänen beider Systeme überschneiden sich dabei stark. Vorgehen und Ergebnisse dieser Fallstudie 3 sind in Abbildung 59 dargestellt.

Im ersten Schritt wird durch die Gewichtung der Kriterien ein Anforderungsprofil definiert. Hierbei gelten bei dieser Fallstudie dieselben technischen, betrieblichen und organisatorischen Rahmenbedingungen, wie bei Fallstudie 2. Es ergibt sich ein Optionsraum aus 72 möglichen Archetypen von Integrationsarchitekturen, aus welchem sechs als geeignet empfohlen werden. Die Ergebnisse des GAP-Modell ergeben dabei die Empfehlung das Storage-Driven-Paradigma in einer Tree-Topologie umzusetzen und die Systeme mit einem Wrapper für webbasierte Integrationsmethoden zu ertüchtigen.

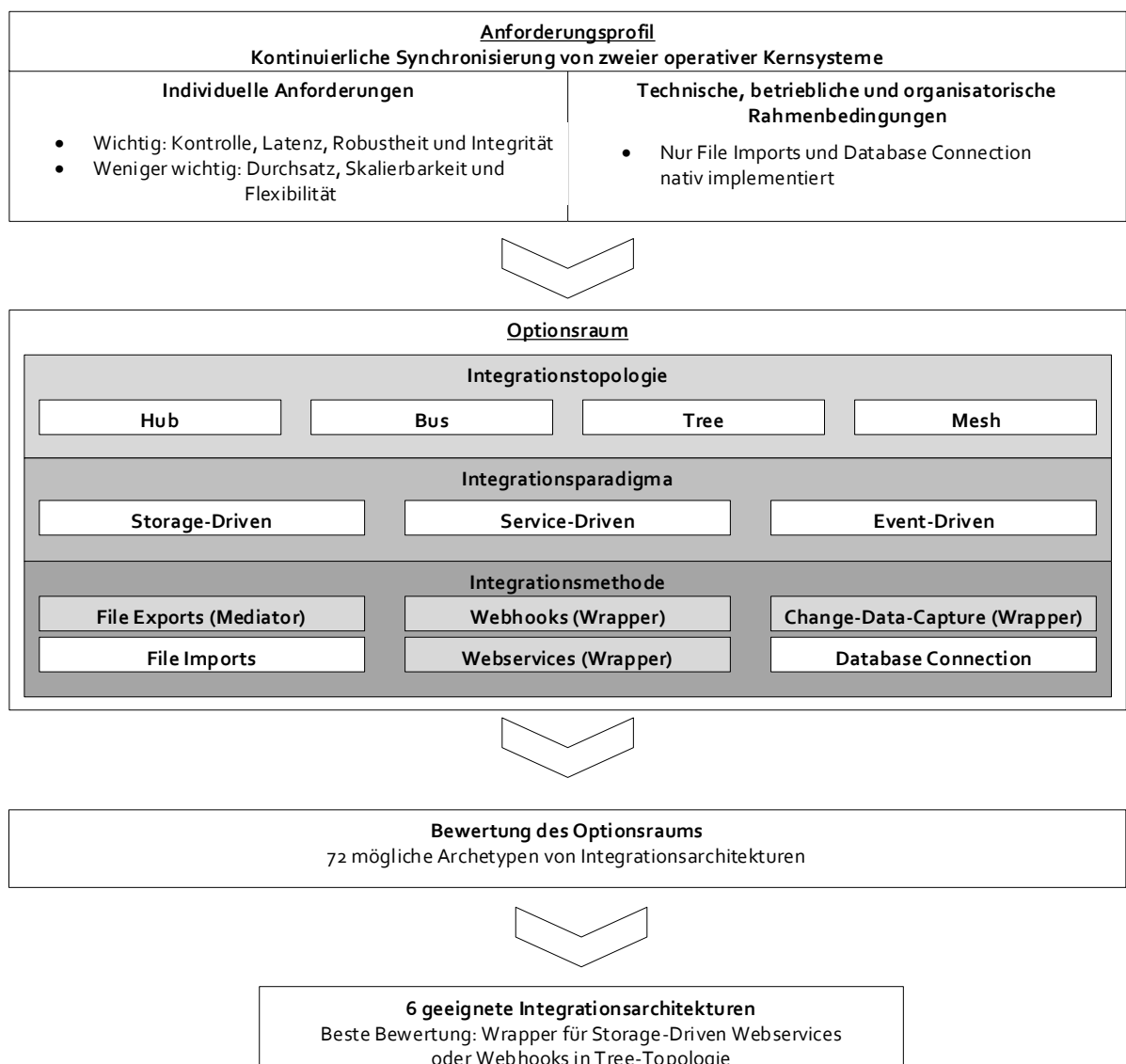


Abbildung 59: Vorgehen und Ergebnisse von Fallstudie 3

6.4.1. Beschreibung der Anforderungen

In folgendem werden die Anforderungen die Integrationsarchitektur zur Synchronisierung zweier operativer Kernsysteme beschrieben. Hintergrund für diese Fallstudie ist die Anforderung, dass ein Bestandssystem in einen Übergangszeitraum im Parallelbetrieb mit einem neuen Produkt genutzt werden soll. Das zu ersetzende Bestandssystem wurde bereits als Fallstudie für Satellitenintegration in vorherigen Abschnitt näher beleuchtet. Es handelt sich um ein operatives System zur Planung und Durchführung klinischer Studien. Abbildung 6o zeigt das individuelle Anforderungsprofil mit gewichteten Kriterien.

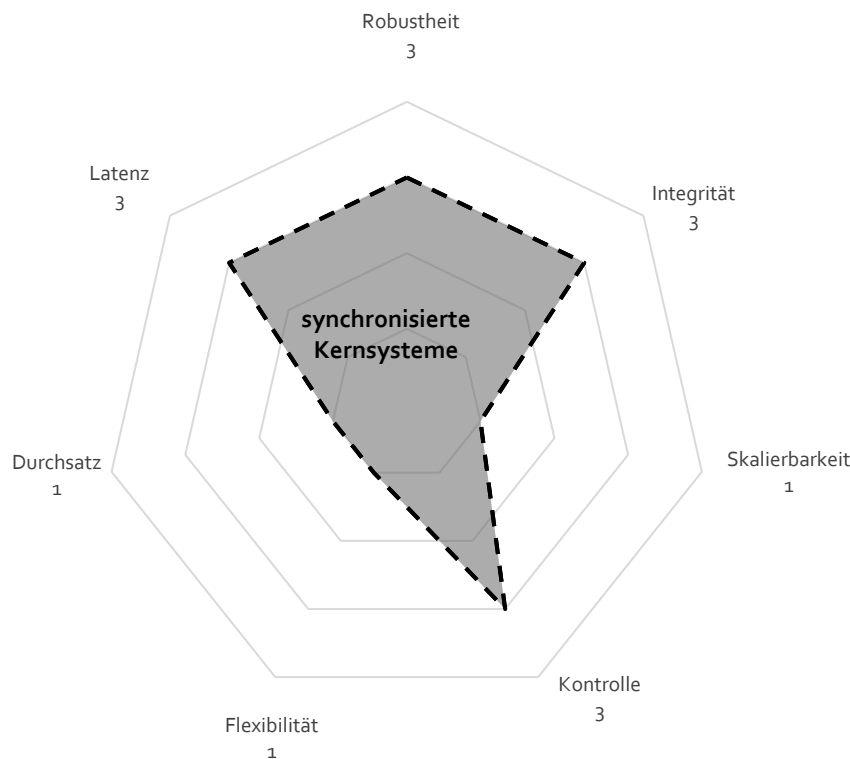


Abbildung 6o: Anforderungsprofil an Integrationen zur Synchronisierung operativer Kernsysteme

Geschäftsprozesse und Funktionalitäten sind parallel implementiert, so dass beide auf dieselbe Datenbasis zugreifen. Änderungen in einem der Systeme müssen dabei in alle synchronisierten Systeme übertragen werden. Ausgewählte (Teil-)Prozesse und Funktionen werden dabei in einem System durchgeführt, wobei andere wiederum in einem anderen System umgesetzt sind. Der Parallelbetrieb ist nötig, um die operative Integrität zu jeder Zeit zu gewährleisten und trotzdem einen Systemwechsel zu implementieren. Derzeit sind noch nicht alle bestehenden Geschäftsprozesse im neuen System konfiguriert. Neue Anforderungen werden jedoch auch nicht mehr im Altsystem umgesetzt. Zudem ist aus regulatorischer Sicht ein Systemwechsel bei laufenden Studien schwierig, so dass auf eine Migration von zeitlich bereits stark fortgeschrittenen Studien verzichtet werden soll. Diese Anforderungen beziehen sich auf den speziellen Anwendungsfall, lassen sich jedoch auch auf kontinuierliche parallel genutzte Systeme übertragen, bei denen beispielsweise Masterdaten parallel in beiden Systemen gepflegt werden.

Wie in Abbildung 6o dargestellt, ergibt sich eine hohe Anforderung an *Latenz*, *Robustheit*, *Integrität* und *Kontrolle* der Systemintegration. Passende Integrationsarchitekturen müssen in diesen Kriterien mindestens als *stark* (3) bewertet sein. Die Kriterien *Flexibilität*, *Durchsatz* und

Skalierbarkeit stehen bei der Auswahl der Integrationsarchitektur nicht im Fokus und können somit auch als *sehr schwach* (1) bewertet sein.

Die Anforderungen nach geringer *Latenz* und hoher *Integrität* der Integration zwischen den synchronisierten Systemen stehen in enger Wechselwirkung und ergeben sich daraus, dass in beide Systemen dieselben Entitäten gepflegt werden.

Eine hohe *Integrität* bedeutet, dass es keine Datenkonflikte gibt und jeder Akteur im Kommunikationsökosystem jederzeit über die korrekten Daten verfügt, die Systeme also synchronisiert sind.

Eine hohe *Latenz* bedeutet, dass Daten nur mit zeitlichem Verzug synchronisiert werden. In dieser zeitlichen Lücke besteht Asynchronität und es kann zu Datenkonflikten kommen. So können beispielsweise Prozesse anhand veralteter Daten falsch angestoßen werden. Auch könnte die Integration veralteter Daten zwischenzeitliche Datenänderungen in einem anderen System überschreiben. Wenn mehrere Systeme dieselben Datenobjekte Nutzen, muss also dafür Sorge getragen werden, dass entweder keine parallele Nutzung der Daten in mehreren Systemen stattfindet oder das Zeitfenster zwischen den Synchronisierungen so klein ist, dass es Datenkonflikte im tatsächlichen Betrieb vermieden werden. Lösungsansätze hierfür sind beispielsweise eine Echtzeit-Integration mit push-Pattern, oder eine Virtualisierung der Daten bei der es nur ein Hauptsystem gibt, auf welches alle anderen Systeme referenzieren.

Die Aspekte *Robustheit* und *Kontrolle* sind bei der Auswahl der Integrationsarchitektur sehr wichtig. Es handelt sich bei den Kernsystemen wie bereits in Abschnitt 6.3.1 beschrieben um validierte Computersysteme in einem GxP-relevanten Umfeld. Da es sich bei den Kernsystemen um geschäftskritische Applikationen handelt, müssen diese hochverfügbar und sicher sein.

Robuste Integrationen zeichnen sich dadurch aus, dass Consumer im Konnektivitätsökosystem auch dann noch funktionieren, wenn die Provider ausgefallen sind. Im Konkreten Anwendungsfall bedeutet dies, dass der Wegfall eines parallel betriebenen Systems keine Auswirkungen auf die Funktionstüchtigkeit des anderen synchronisierten Systems haben darf.

Die *Kontrolle* über die Integrationen ist im betrachteten Anwendungsfall besonders wichtig, da unerkannte Fehler oder Manipulationen bei der Integration die operative Datenbasis korrumpieren könnten. So könnte beispielsweise ein Implementierungsfehler zu falschen Daten im System führen. Auch könnte durch ausbleibende Synchronisierung die Integrität der Daten untergraben werden, weil parallel anhand der nun asynchronen Datensätze in beiden Systemen Prozesse falsch angestoßen werden könnten. Schlechte Kontrolle über die Integrationen stellt also ein großes Risiko für die Integrität des validierten Computersystems dar und muss ausgeräumt werden.

Das Kriterium *Durchsatz* erfährt im betrachteten Anwendungsfall keine hohe Priorität. Es ist davon auszugehen, dass bei einer kontinuierlichen Synchronisierung keine großen Datenmengen ausgetauscht werden müssen. Wenn Änderungen also immer unverzüglich übertragen werden, kann der Durchsatz im laufenden Betrieb bei der Integration ignoriert werden, wenn beide Systeme durch eine initiale Synchronisierung auf demselben Stand sind. Im Sinne der Integrität könnte jedoch beispielsweise ein täglicher Datenabgleich anhand eines „Full-Loads“ erfolgen, welcher jedoch in einer Zeit niedriger Auslastung geplant werden kann, so dass auch hier der Durchsatz zweitrangig ist.

Flexibilität und *Skalierbarkeit* sind im Anforderungsprofil als unwichtig bewertet, da es sich um eine temporäre Lösung handelt, bei welcher keine starken Änderungen, oder kontinuierlichen Anpassungen erforderlich sein werden. Nichtsdestotrotz können diese Aspekte Relevanz erlangen, wenn nach einer langfristigen Lösung für die kontinuierliche Synchronisierung in einem dynamischeren Konnektivitätsökosystem gesucht wird.

6.4.2. Anwendung des GAP-Modells

Tabelle 15 zeigt die Bewertungsergebnisse für die sechs vom GAP-Modell als geeignet integrierten Architekturen identifizierte Kombination von Designartefakten. Es werden pro Zeile die Wertpunkte der entsprechenden Integrationsarchitektur für jedes Kriterium, sowie eine allgemeine Bewertung gezeigt. Alle Architekturarchetypen erfüllen die Anforderungen an Robustheit, Integrität, Kontrolle und Latenz, wobei auch alle Integrationsarchitekturen stark in Bezug auf Skalierbarkeit und Durchsatz sind. Bei der Flexibilität zeichnet sich ein gemischtes Bild, wobei dieses Kriterium für die Bewertung unerheblich ist. Eine detaillierte Bewertung des gesamten Optionsraums finden sich in Anhang 14 ab Seite 166.

				Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Anforderungsprofil				3	3	1	3	1	1	3	
Architekturarchetypen	Tree	Storage-Driven	Webservices	3	3	4	3	2	3	3	7,64
	Tree	Storage-Driven	Webhooks	3	3	4	3	2	3	3	7,64
	Tree	Storage-Driven	Database-Connection	3	3	4	3	1	4	3	7,45
	Bus	Storage-Driven	Database-Connection	3	3	3	3	2	4	3	7,33
	Tree	Storage-Driven	Change-Data-Capture	3	3	4	3	1	4	3	7,26
	Tree	Service-Driven	File-Imports	3	3	4	3	3	3	3	7,17

Tabelle 15: Bewertungsergebnisse des GAP-Modells für Integrationen zur Synchronisierung operativer Kernsysteme

Das GAP-Modell schlägt für den betrachteten Anwendungsfall das physische Kopieren der Daten mithilfe einer webbasierten Integration in der direkten Interaktion zwischen den integrierten Systemen vor. Datenbankbasierte Integrationsmethoden erreichen hier mit Tree-Topologie nur die Mindestbewertung, wobei webbasierte und dateibasierte Integrationen durch die Entkopplung mehr Flexibilität in der Zusammensetzung des Kommunikationsökosystems erlauben.

Abbildung 61 zeigt die drei nach GAP-Modell am besten bewerteten Optionen für die Integration der synchronisierten Kernsysteme im Vergleich zum Anforderungsprofil. Storage-Driven Webservices, sowie Webhooks im Integration-Tree erreichen dieselbe Punktzahl, weshalb diese hier als Option A zusammengefasst sind. Integrationstopologien mit exklusiver Kommunikationsflexibilität (Integration-Tree, und -Bus) erzielen die besten Ergebnisse. In Bezug auf das Integrationsparadigma scheint insbesondere für die hohe Anforderung an Robustheit eine Storage-Driven-Integration am geeignetsten.

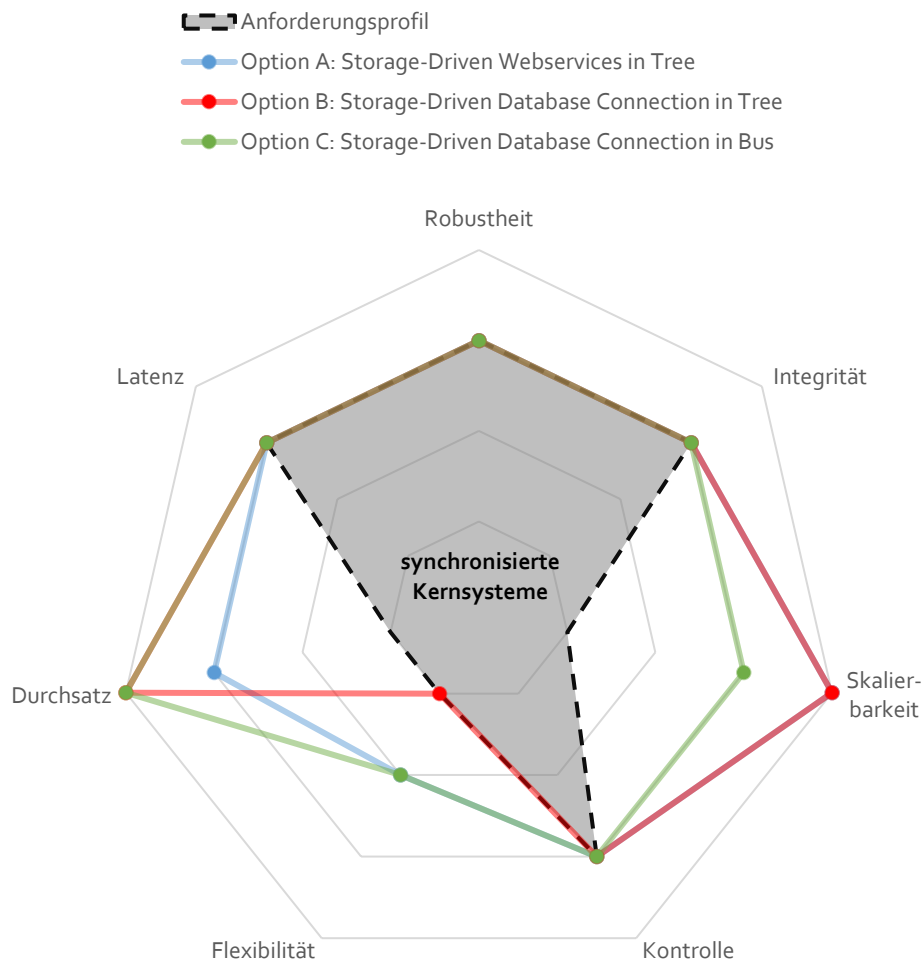


Abbildung 61: Integrationsarchitekturen für Fallstudie 3 mit verschiedenen Topologien und Methoden im Vergleich

Als Integrationstopologien scheinen sowohl Integration-Tree als auch Integration-Bus geeignet. Bei beiden Topologien werden die Daten direkt zwischen den Systemen miteinander geteilt. Beim Integration-Tree erfolgt dieses im direkten Austausch, beim Integration-Bus kommt hierfür eine gemeinsam genutzte Middleware zum Einsatz. Beide Integrationstopologien zeichnen sich durch Stärke beim Kriterium der Kontrolle aus, da die Integrationspfade klar und direkt sind. Ein Integration-Hub scheint aufgrund der Nachteile in Bezug auf Robustheit ungeeignet und ein Integration-Mesh führt zu Herausforderungen bei der Integrität.

Für die Auswahl des Paradigmas zeigen die Ergebnisse des GAP-Modells eine klare Favorisierung von Storage-Driven-Integrations. Dies ergibt sich insbesondere aus der Stärke dieses Integrationsmusters bei Robustheit und Kontrolle. Schwächen bei der Flexibilität fallen aufgrund des Anforderungsprofils nicht ins Gewicht und mögliche Nachteile bei Latenz und Integrität werden durch die Stärken der Integrationsmethoden in diesen Bereichen kompensiert. Auffällig ist, dass keine der empfohlenen Integrationen dem Event-Driven-Paradigma folgt, obwohl dieses sich durch Stärken in Bezug auf Latenz und Integrität auszeichnet. Insbesondere die Schwäche in Bezug auf Kontrolle haben hier eine Passung zum Anforderungsprofil im GAP-Modell verhindert.

Es werden verschiedene Integrationsmethoden vorgeschlagen, wobei diese jeweils die topologie- und paradigmabedingte Schwächen ausgleichen. Webbasierte Integrationen sind sehr stark in Bezug auf Integrität, Kontrolle und Latenz, sind jedoch weniger robust als andere Integrationsmethoden. Diese Schwäche kann beispielsweise durch das Storage-Driven-Paradigma sehr gut aufgefangen werden, indem Daten einfach kopiert werden und so unabhängig von der Verfügbarkeit des Provider-Systems immer verfügbar sind. Auch Datenbankverbindungen sind sehr stark in Bezug auf Integrität und Latenz, haben aber Nachteile bei der Robustheit und der Kontrolle. Genau wie bei webbasierten Integrationen werden diese Nachteile durch die Wahl der richtigen Integrationstopologie und eines geeigneten Paradigmas ausgeglichen.

Zusammenfassend ergibt sich aus den Ergebnissen des GAP-Modells eine Integrationsarchitektur, bei der Daten physisch direkt zwischen allen synchronisierten Systemen kopiert wird. Dabei ist unerheblich, ob der Datenaustausch auf Datenbankebene implementiert ist, die Integration mit einer webbasierten API gekapselt wird, oder ein dateibasierter Austausch stattfindet. Wichtig ist nur, dass die Integration möglichst robust implementiert wird und die Integration entweder kontinuierlich in sehr kurzen Abständen abläuft, oder durch Änderungen in einem System proaktiv ausgelöst wird.

6.4.3. Beschreibung der tatsächlichen Implementierung

Beim betrachteten Anwendungsfall handelt es sich um eine Synchronisierung zum Parallelbetrieb zweier Clinical Trial Management Systeme, wobei System A ein etabliertes Altsystem ist und Stück für Stück vom moderneren System B ersetzt werden soll. Die Integration wurde dabei in zwei Phasen umgesetzt. In der Anfangsphase, bestand noch keine Funktionsüberlappung, da mit System B nur neue Funktionalität eingeführt wurde, ohne Funktionalität aus System A zu ersetzen. In dieser Anfangsphase wurde eine einfache Synchronisation der Stammdaten mithilfe eines täglichen File-Import über einen Integration-Hub umgesetzt (Implementierung 1). Mit eintreten dem Parallelbetrieb und höherer Reife des Anwendungsfalls wurde die Integrationsmethode auf Webservices umgestellt, um die Synchronisierungslücke zu minimieren (Implementierung 2). Zusätzlich wurde prototypisch eine eventbasierte Integration (Implementierung 3) implementiert, welche jedoch aufgrund von Stabilitätsproblemen verworfen wurde. Diese Integrationen sind in Abbildung 62 schematisch dargestellt.

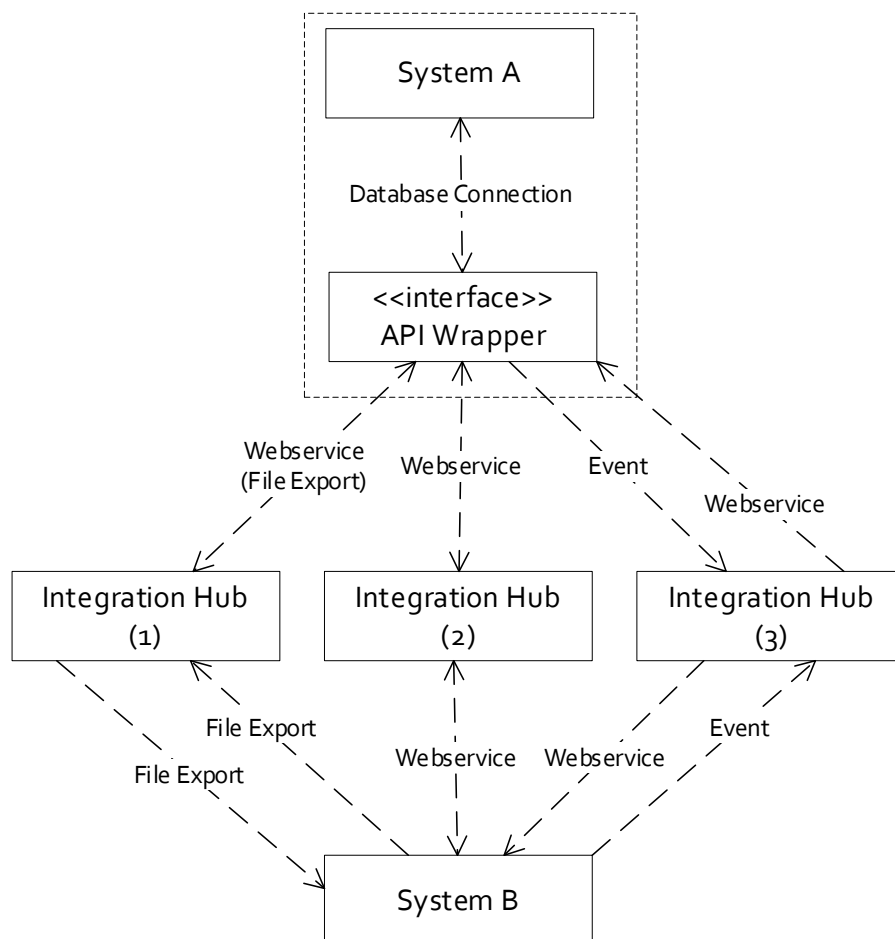


Abbildung 62: schematische Darstellung tatsächlich implementierter Integrationen in Fallstudie 3

Synchronisierung mit File-Exports (Implementierung 1)

Bei der zuerst implementierten Synchronisierung mit Integration-Hub handelt es sich um die Umsetzung eines klassischen ETL-Prozesses. Wie oben beschrieben, konnten System A und System B in der ersten Phase noch funktional getrennt voneinander betrieben werden, so dass ein Datenaustausch mit hoher Latenz zunächst genügte. Dabei wurde eine Implementierung gewählt, welche einfach und aufwandsarm umgesetzt werden konnte.

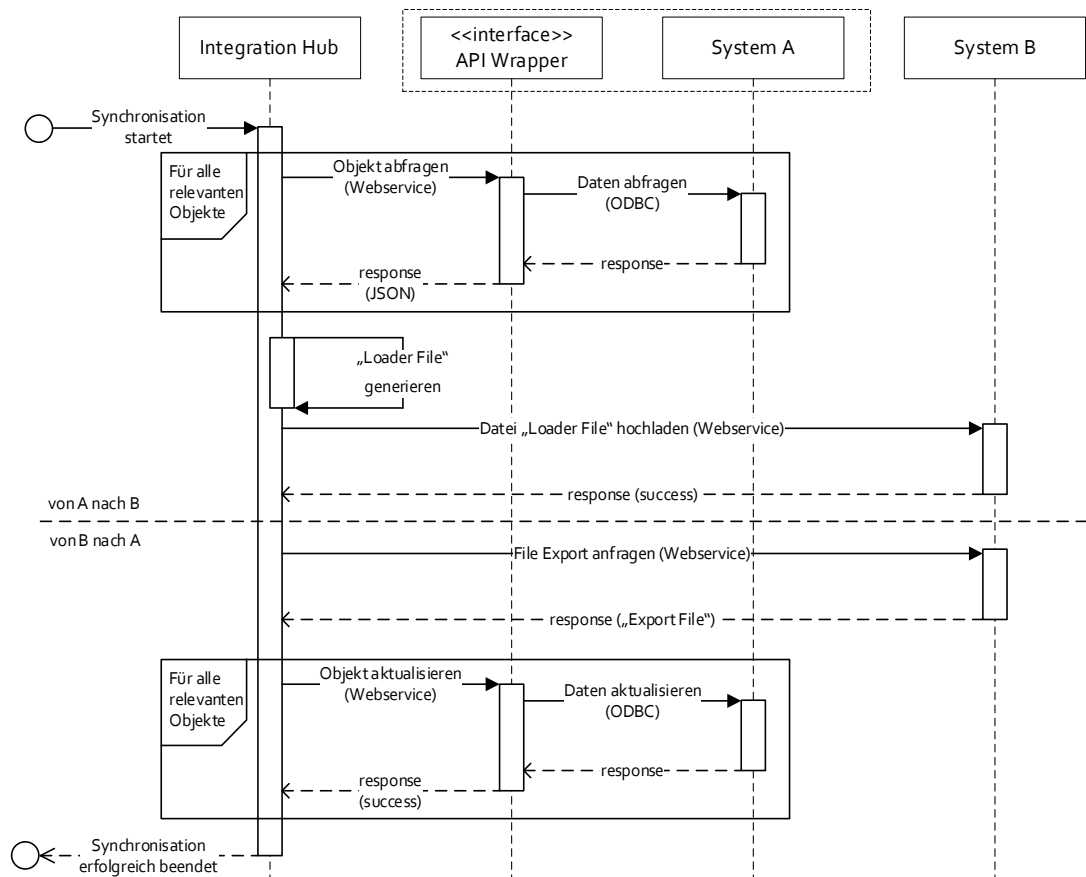


Abbildung 63: Ablauf der Synchronisierung mit File-Exports in Fallstudie 3

Die Implementierung 1 kann im GAP-Modell als Storage-Driven-Integration mit File-Exports in einem Integration-Hub kategorisiert werden. Wie in Abbildung 63 dargestellt, erfolgt die Synchronisation in zwei Phasen. Jegliche Kommunikation erfolgt dabei ausgehend vom Integration-Hub, welcher den Integrationsablauf viermal täglich initiiert.

Zuerst werden alle relevanten Daten über die nachträglich implementierte Web-API abgefragt und zu einer Datei zusammengefasst. Dieses „Loader File“ wird dann über einen Webservice in System B hochgeladen, wo die Daten physisch abgelegt werden.

Die Synchronisierung von System B nach System A erfolgt analog auf dem entgegengesetzten weg. Die Daten werden als File-Export von System B angefragt und dann über den API-Wrapper in System A übertragen.

Obwohl die Kommunikation auch über einen Webservice realisiert wird, ordnet der Autor für diese Implementierung im GAP-Modell die Integrationsmethode der Kategorie File-Export zu. Der Datenaustausch ist im Wesentlichen dateibasiert realisiert und erfolgt vom Integration-Hub

ausgehend im pull-Pattern an System B. Auch der Datenaustausch von System B zurück nach System A erfolgt mit File-Exports.

Die Synchronisierung der operativen Kernsysteme mit File-Exports über einen Integration-Hub hat sich in der Praxis als schnelle pragmatische Lösung bewährt, ist jedoch bei veränderten Rahmenbedingungen schnell an konzeptionelle Grenzen gestoßen. Zudem hat sich gezeigt, dass die Middleware des API-Wrappers zwischenzeitlich den Anforderungen an den Datenexport nicht gewachsen war, so dass Anfangs vermehrt Stabilitätsprobleme bei der Integration zu beobachten waren. Erst durch Änderungen am Design des Webservices und Optimierung der zugrundeliegenden Abfragen konnte die Integration stabil betrieben werden.

Wie erwartet, war die Integration mit File-Exports auch nur so lange geeignet, wie keine parallelen Aktivitäten an denselben Objekten in beiden Systemen erforderlich waren. Unter den gegebenen technischen Rahmenbedingungen erwies sich die Erstellung von File-Exports über die existierenden Webservices und das Hochladen der „Loader Files“ als so zeitaufwändig, dass eine engere Taktung als ein zweistündlicher Datenaustausch nicht realisiert werden konnte. Für einen wirklichen Parallelbetrieb müssten die Zeitfenster viel kleiner sein, da es ansonsten schnell zu Dateninkonsistenzen kommen könnte. In der Praxis wurden entsprechende Felder in den jeweiligen Systemen zunächst schreibgeschützt und Funktionen deaktiviert.

Ein weiteres Problem, welches sich in der Praxis bei dieser Implementierung gezeigt hat, ist die Fehleranfälligkeit dateibasierter Integrationen. Unvorhergesehene Strukturen und Formatierungen in den Ausgangsdaten führten zu Problemen beim Generieren, oder Interpretieren der „Loader Files“, so dass ein intensives Monitoring der Integrationen im laufenden Betrieb von Nöten war. Hierbei hat sich die Entscheidung für einen Integration-Hub als sehr gut erwiesen, da so zu jeder Zeit volle Kontrolle über die Integrationen herrscht und eventuelle Fehler so schnell identifiziert und behoben werden können.

Synchronisierung mit Webservices (Implementierung 2)

Bei der hier beschriebenen Implementierung 2 handelt es sich um eine Weiterentwicklung der Implementierung 1 mit veränderter Integrationsmethode. Die Daten werden dabei nun nicht mehr täglich „en bloc“ in einem konsolidierten „Loader File“ für alle Objekte übertragen, sondern kontinuierlich und kleinteilig pro Objekt mithilfe von Webservices synchron gehalten. Ziel war, die Latenz mit den verfügbaren technischen Fähigkeiten der Systeme möglichst gering zu halten und zugleich durch Entkopplung und Modularisierung die Flexibilität für spätere Anpassungen im Zuge des Migrationsprozesses zu schaffen.

Die Implementierung 2 ist im GAP-Modell dem Archetypen Storage-Driven-Integration mit Webservices in einem Integration-Hub zugeordnet. Wie in Abbildung 64 dargestellt, erfolgt eine kontinuierliche Synchronisation zwischen beiden Systemen für alle relevanten Objekte mithilfe von Webservices.

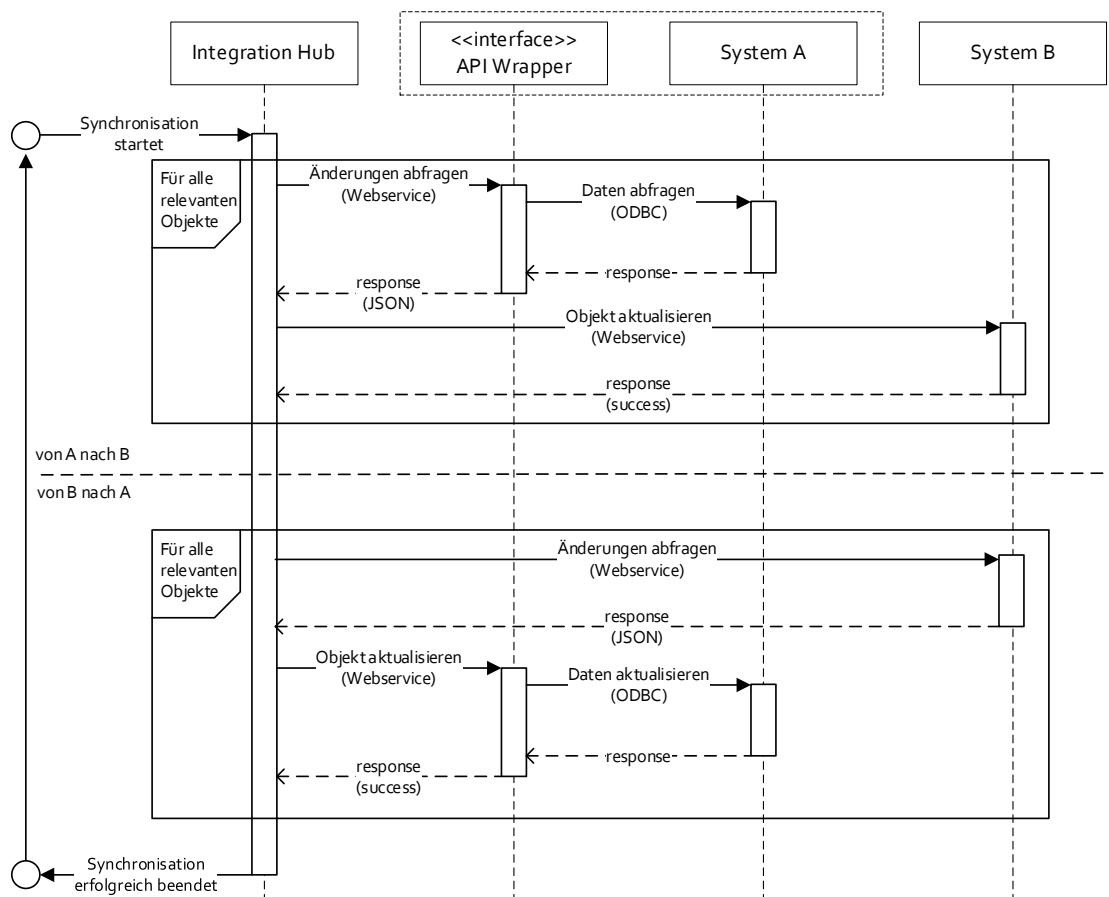


Abbildung 64: Ablauf der Synchronisierung mit Webservices in Fallstudie 3

Ausgehend vom Integration-Hub wird der API-Wrapper von System A angefragt, ob es Änderungen an relevanten Objekten gab. Diese Antwort wird anhand Daten in der Datenbank von System A sofort gegeben und vom Integration-Hub verarbeitet. Wurden Änderungen an einem Objekt vorgenommen, so wird das korrespondierende Objekt in System B punktgenau über einen Webservice aktualisiert. Anschließend wird nach gleichem Muster in System B über einen Webservice nach Änderungen an relevanten Objekten angefragt, was wiederum eine Aktualisierung über vom API-Wrapper von System A bereitgestellte Webservices initiiert. Anders als bei der dateibasierten Integration in Implementierung 1 muss die Synchronisierung nicht zwangsläufig auf Daten-Ebene erfolgen, sondern kann über die Webservices abstrahiert werden. Dies ist vor allem

relevant, wenn die Datenstrukturen beider Systeme konzeptionell nicht exakt übereinstimmen. Ein Beispiel hierfür ist die Zuordnung von Personen zu Rollen, oder die Synchronisation von Plan- und Meilensteinen.

Aufgrund von Abhängigkeiten zwischen bestimmten Objekten wurden einige Objekte bei der Synchronisierung in einen Ablauf zusammengefasst. Auch wird im Integration-Hub geprüft, ob Datenkonflikte bestehen. Können diese anhand einfacher Regeln, beispielsweise anhand von Zeitstempeln, nicht gelöst werden, so werden die Änderungen nicht übertragen und zu einem späteren Zeitpunkt manuell ausgeräumt. Da in der Praxis die Synchronisationszeiträume im unteren einstelligen Minutenbereich lagen, sind solche Datenkollisionen jedoch äußerst selten und stellen keine Einschränkung der operativen Integrität dar.

Die tatsächliche Implementierung hat sich als gut erwiesen. Beide Systeme befinden sich im stabilen Parallelbetrieb, was insbesondere durch die geringe Latenz und hohe Integrität der Integration ermöglicht wurde. Großer Nachteil dieser Implementierung ist, dass die komplette Integration über eine zentrale Komponente realisiert ist. Ein Ausfall des Integration-Hubs bedeutet einen Ausfall der Synchronisierung. Bleibt dieser Zustand unerkannt, so werden Änderungen nicht mehr zwischen den Systemen ausgetauscht und es entstehen zwei „Datenrealitäten“. In diesem Fall muss der Betrieb mindestens eines Systems unterbrochen werden, bis die Integration wiederhergestellt ist und die Datenbasis konsolidiert werden konnte. Anderenfalls könnten anhand falscher Daten Prozesse fehlerhaft ausgeführt werden und die Validität des Computersystem gefährden (GxP Compliance).

Synchronisierung mit Events (Implementierung 3)

Die in Abbildung 65 dargestellte Integrationsarchitektur wurde prototypisch implementiert, um die Machbarkeit einer Echtzeit-Integration bei völliger Entkopplung von vorgegebenen Systemstrukturen zu untersuchen. Hierbei kann theoretisch jedes System dynamisch als Consumer oder Provider in Konnektivitätsökosystem ein- oder ausgehängt werden, indem dieses entweder Events sendet oder auf diese reagiert.

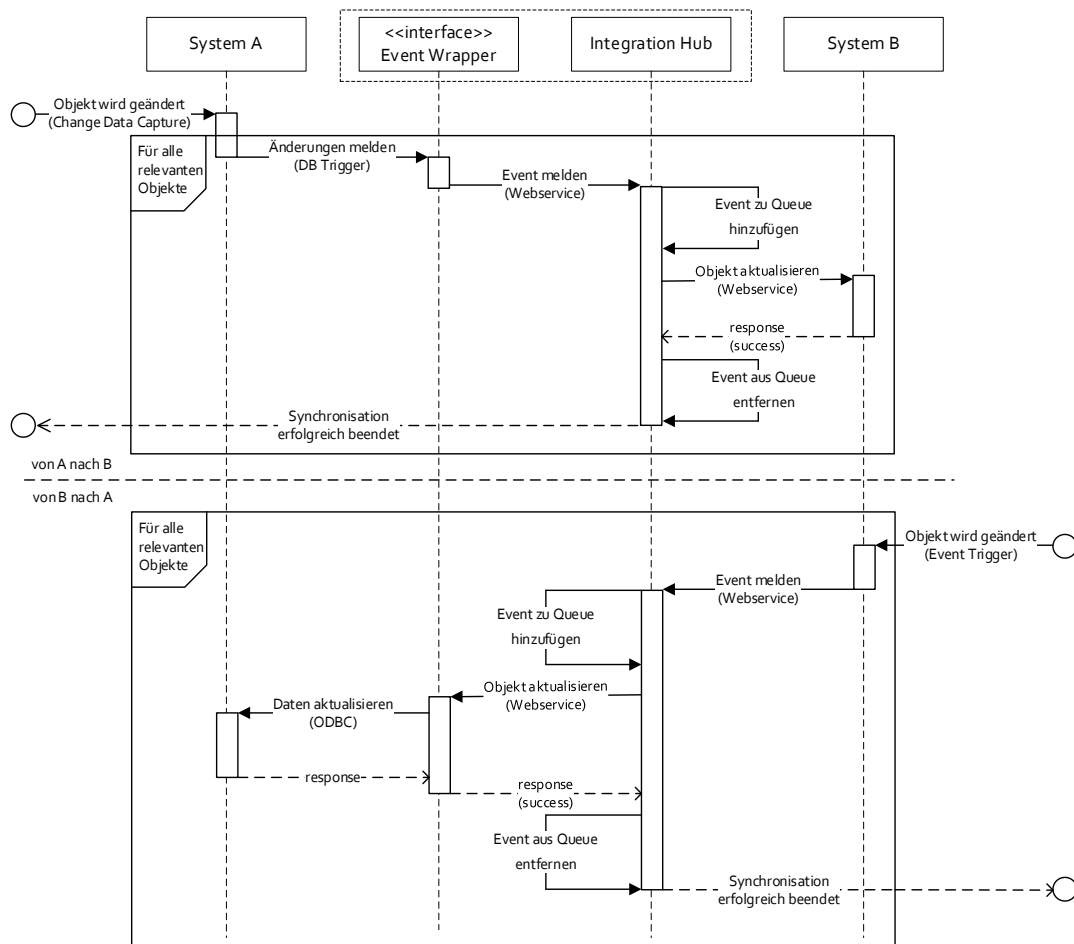


Abbildung 65: Ablauf der Synchronisierung mit Events in Fallstudie 3

Jedes teilnehmende System kommuniziert Änderungen an den Daten über einen HTTP-Endpunkt an einen „Notification Service“, welcher dann darauf basierende vorher definierte Aktivitäten ausführt. Zur besseren Kontrolle und Robustheit werden alle Events in einer Queue gespeichert. Im konkreten Anwendungsfall wird bei Objektänderungen ein Event ausgelöst, woraufhin der Integration-Hub dann sofort die Änderungen über Webservices an das andere System überträgt.

Eine Event-Driven Integration hat sich in der Praxis als nicht zweckdienlich erwiesen, da für eine robuste Integration sichergestellt werden müsste, dass die integrierten Systeme ausnahmslos bei allen Änderungen ein Event schicken, dass dieses immer korrekt empfangen und verarbeitet wird und dass alle Komponenten immer verfügbar sind. Im Vergleich zu Implementierung 2 konnte die Latenz weiter auf wenige Millisekunden verbessert werden, so dass Datenkonflikte praktisch nicht mehr möglich wären. Durch die konsequente Entkopplung konnte zudem maximale Flexibilität erreicht werden. Dem entgegen steht jedoch ein deutlicher Verlust an Robustheit und gesteigerter Komplexität der Implementierung und Infrastruktur. Zudem ist dieses

Integrationsparadigma bei den betrachteten Systemen noch nicht etabliert und derzeit ausschließlich für sehr spezielle Anwendungsfälle konzipiert, bei denen beispielsweise aufgrund einer Dateneingabe Produktionsprozesse gestartet werden sollen. Eine Integration mit Webservices ließ sich zudem besser automatisieren, da beide zu integrierenden Systeme hier über ausführliche Endpunkte mit Metainformationen verfügen.

Letztendlich wurde dieses Konzept der eventgetriebenen Integration nicht weiterverfolgt und wurde somit auch nicht beim Application Assessment beachtet. Die Bewertung nach GAP-Modell findet sich im folgenden Abschnitt und deckt sich mit den beschriebenen Beobachtungen aus der Praxis.

6.4.4. Vergleich mit den Ergebnissen des Application Assessments

Abbildung 66 zeigt die Bewertungsergebnisse des GAP-Modells für die Integrationsarchitekturen, welche zur kontinuierlichen Synchronisierung operativer Kernsysteme implementiert wurden. Verglichen werden diese mit dem Anforderungsprofil, sowie der vorgeschlagenen Integrationsarchitektur.

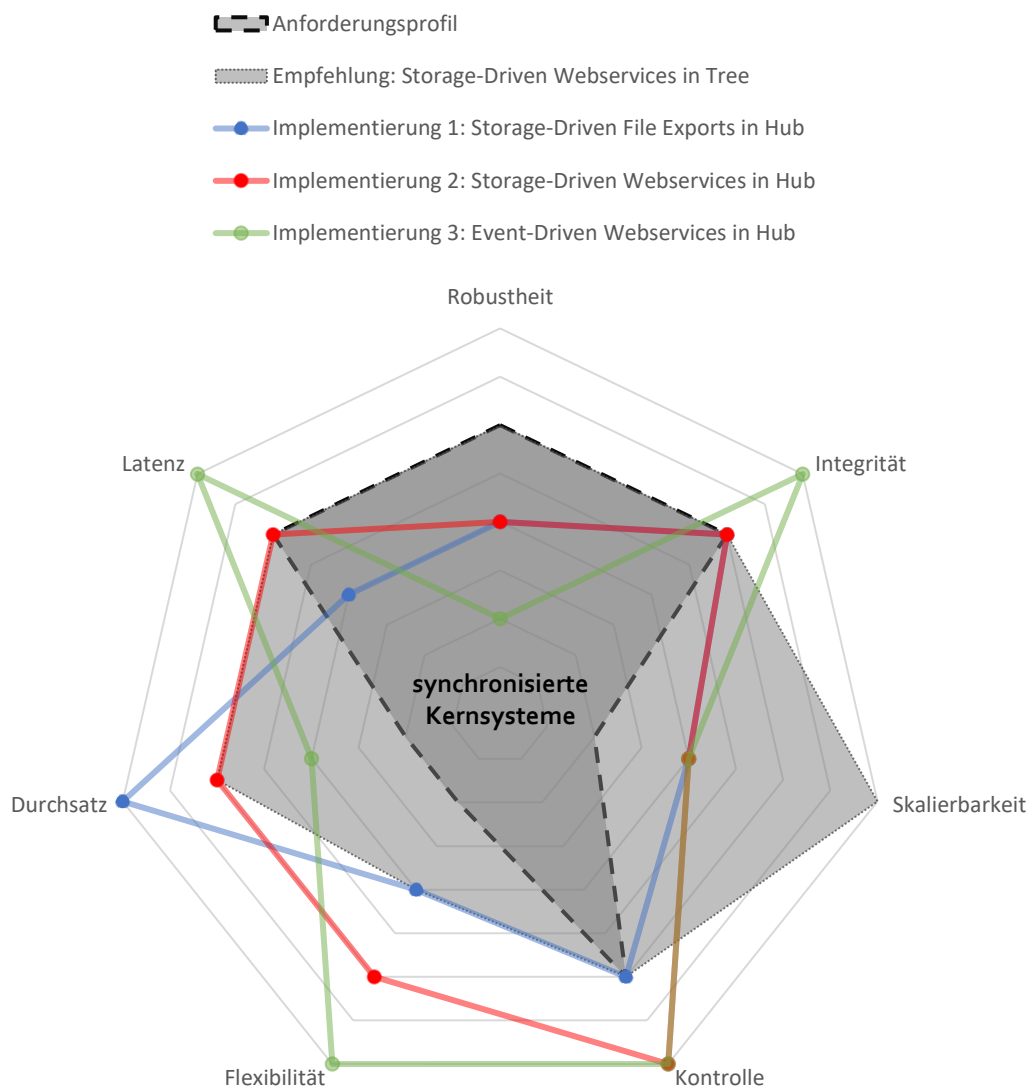


Abbildung 66: Ergebnisse für Integrationen zur kontinuierlichen Synchronisierung operativer Kernsysteme

Wie in vorherigen Abschnitt beschrieben, wurden im betrachteten Anwendungsfall drei verschiedene Integrationsarchitekturen implementiert. Alle drei implementierten Integrationen folgen einer Hub-Topologie. Implementierung 1 und 2 folgen dem Storage-Driven-Paradigma, wohingegen mit der prototypischen Implementierung 3 ein Event-Drive-Paradigma verfolgt wurde.

Im GAP-Modell erfüllt die Implementierung 1 bei allen Kriterien außer Robustheit und Latenz die Anforderungen und erreicht beim Durchsatz einen Spitzenwert. Implementierung 2 liegt als

Weiterentwicklung nur noch bei der Robustheit knapp unter den Anforderungen und zeichnet sich durch verbesserte Latenz und Flexibilität, sowie eine sehr starke Kontrolle aus. Die prototypische Implementierung 3 erreicht eine sehr starke Bewertung bei Latenz, Integrität, Kontrolle und Flexibilität, schneidet jedoch bei der Robustheit sehr schlecht ab.

Diese Bewertung stimmt mit den Erfahrungen bei der tatsächlichen Implementierung überein, da der Prototyp aufgrund konzeptionell begründeter Einschränkungen bei der Robustheit der Integration nicht weiter entwickelt wurde.

Für die beiden produktiv genutzten Integrationen (Implementierung 1 und 2) wurde im Rahmen des Applikationsmanagements ein Application Assessment nach TOGAF durchgeführt. Die Ergebnisse dieser Analyse werden in Folgendem mit den Ergebnissen des GAP-Modells verglichen. Für die Implementierung 3 liegen keine unabhängigen Bewertungsergebnisse vor, da es sich lediglich um eine prototypische Integration handelt.

Synchronisierung mit File-Exports (Implementierung 1)

Abbildung 67 zeigt die Ergebnisse des Application Assessments für die übergangsmäßig implementierte Integration mit File-Exports. Bei keiner der relevanten Kategorien ist die Implementierung schlecht bewertet.

The screenshot displays an 'Application Assessment' form for 'synchronisiertes Kernsystem (File Exports)'. The form is organized into three columns of criteria, each with a rating or status.

Category	Rating/Status	Category	Rating/Status	Category	Rating/Status
Application *	synchronisiertes Kernsystem (File Exports)	Off-the-shelf	Near-complete Coverage / Acceptat	API First	Near-complete Coverage / Acceptat
Modular Architecture	Near-complete Coverage / Acceptat	Security and Compliance	Near-complete Coverage / Acceptat	Data assets/products	Clinical Studies, Study Countries, Study Sites
Training and documentation	—	Operational Excellence	Minor Gap in Coverage	Decommissioned date	Enter value here
Replaced by	Enter value here	Plan for Decommission	—	Tech Fit for future	Med
URL to the application architecture	Enter value here	BeatID	BEAT04006402	Tech Fit for now	high
Business fit for now	—	Business fit for future	high	data follow pattern	Data Prosumer
Hosting type	SaaS	Clinical Development Process	Design, Setup, Conduct, Analyze	Development function	—

Abbildung 67: Ergebnisse des Application Assessment für Integration mit File-Exports
(Quelle: Eigener Screenshot aus einem internen Expertensystem)

Die Integration wird als akzeptabel in Bezug auf Sicherheit („Security and Compliance“) bewertet und erhält eine etwas schlechtere Bewertung („Minor Gap in Coverage“ - Anforderungen verfehlt) in Bezug auf Stabilität, bzw. „Operational Excellence“. Diese Einschätzung bestätigt die Einordnung im GAP-Modell, wonach die Integrationsarchitektur den Anforderungen in den Kategorien Integrität und Kontrolle entspricht, bei der Robustheit jedoch unterhalb der Anforderungen liegt.

Wie oben beschrieben, wird mit der tatsächlichen Implementierung eine korrekte Integration umgesetzt, welche jedoch immer wieder bei auftretenden Fehlern im Integrationsprozess unterbrochen wird. Die Bewertung von „Technical Fit for future“ als mittelmäßig („*Medium*“) zeigt zudem, dass die Implementierung zum Zeitpunkt der Bewertung noch den Anforderungen entspricht, einem wirklichen Parallelbetrieb jedoch nicht gewachsen sein wird. Dies bestätigt die Einordnung des GAP-Modells, wo Implementierung 1 bei der Latenz unterhalb der geforderten Bewertung liegt.

„Modular Architecture“ und „API first“ werden im Application Assessment mit „*Near-complete Coverage/Accaptable*“ (Anforderungen nahezu erfüllt) bewertet. Dies zeigt, dass die Integration grundsätzlich flexibel und skalierbar konzeptionalisiert ist. Hier ist tendenziell eine Diskrepanz mit der Einschätzung des GAP-Modells zu beobachten.

Die relativ gute Bewertung von „API first“ steht nicht im Einklang mit der Kategorisierung der Integration als File-Export. Bei der Implementierung 1 wurden Web-APIs zur Kommunikation zwischen Systemen und Integration-Hub und genutzt, so dass dies im Sinne von „API-first“ eine akzeptable Bewertung begründen kann. Tatsächlich wurde jedoch konzeptionell kein Webservice im Sinne der Kategorisierung im GAP-Modells benutzt, da das Übertragungsmedium zwar webbasiert, das Übertragungsformat jedoch dateibasiert ist. Somit wird der Anschein erweckt, es handle sich um eine Integration mit Web-APIs, tatsächlich werden die Daten jedoch als Datei übertragen.

Auch die Bewertung von „Modular Architecture“ als „*Near-complete Coverage/Accaptable*“ (Anforderungen nahezu erfüllt) deckt sich nicht mit den Ergebnissen des GAP-Modells. Die Ergebnisse des GAP-Modells zeigen bei den Kriterien Skalierbarkeit und Flexibilität für Implementierung 1 eher schwache Werte. Diesen Kriterien wurde jedoch nur geringe Wichtigkeit zugewiesen, so dass hier beinahe jede Art der Implementierung eine volle Bewertung erhalten müsste. Die Einschätzung von GAP-Modell und Application Assessment stimmen also in diesem Punkt überein, das Anforderungsprofil als Bezugssystem für die relative Bewertung ist jedoch beim erfolgten Application Assessment nicht vollkommen gleich.

Synchronisierung mit Webservices (Implementierung 2)

Abbildung 68 zeigt die Ergebnisse des Application Assessment für die angepasste Integration. Diese wurde im vorigen Abschnitt zur Beschreibung der tatsächlichen Implementierung als „Synchronisierung mit Webservices“ bezeichnet.

Application * synchronisiertes Kernsystem (Webservices)	☒ Off-the-shelf Near-complete Coverage / Acceptable	☒ API First Complete Coverage / Satisfactory
☒ Modular Architecture Near-complete Coverage / Acceptable	☒ Security and Compliance Near-complete Coverage / Acceptable	☒ Data assets/products Clinical Studies, Study Countries, Study Sites
☒ Training and documentation —	☒ Operational Excellence Complete Coverage / Satisfactory	☒ Decommissioned date Enter value here
☒ Replaced by Enter value here	☒ Plan for Decommission	☒ Tech Fit for future High
☒ URL to the application architecture Enter value here	☒ BeatID BEAT04006402	☒ Tech Fit for now high
☒ Business fit for now —	☒ Business fit for future high	☒ data follow pattern Data Prosumer
☒ Hosting type SaaS	☒ Clinical Development Process Design Setup Conduct Analyze	☒ Development function —

Abbildung 68: Ergebnisse des Application Assessment für Integration mit Webservices
(Quelle: Eigener Screenshot aus einem internen Expertensystem)

Die Bewertung der aktualisierten Integration mit Webservices ist durchweg positiv und deutet auf eine gelungene Implementierung hin. Lediglich für „Modular Architecture“ und „Security and Compliance“ kann minmales Verbesserungspotential ausgemacht werden. Dies kann auf konzeptionelle Einschränkungen mit der Robustheit der Integration begründet werden, da aufgrund des eingesetzten Integration Hubs ein zentrales Element existiert, welches kritisch für den sicheren Betrieb der Integration ist.

In der Kategorie „Operational Excellence“ wird eine Bestbewertung erreicht. Dies korrespondiert mit den Ergebnissen des GAP-Modells für Kontrolle und Integrität, bei denen für Implementierung 2 Höchstwerte erreicht werden. Auch die Lücke bei der „Security and Compliance“ stimmt mit der Einschätzung des GAP-Modells in Bezug auf Robustheit überein. Hier erreicht die Integrationsarchitektur von Implementierung 2 nicht ganz den geforderten Wert.

6.5. Zusammenfassung und Auswertung

In diesem Abschnitt werden die Ergebnisse der Fallstudien zusammengefasst und in Hinblick auf die Plausibilität der Ergebnisse des GAP-Modells ausgewertet. Dabei wird mit dem Ziel der Validierung des GAP-Modells untersucht, ob sich die Ergebnisse des unabhängigen Application Assessments nach TOGAF, sowie Beobachtungen zur tatsächlichen Implementierten Integrationen mit den Ergebnissen des GAP-Modells übereinstimmen.

6.5.1. Fallstudie 1: Anbindung verschiedener Systeme an ein operatives Data Warehouse

Für die in Abschnitt 6.2 beschriebene Fallstudie 1 (operatives Data Warehouse) wird die tatsächliche Implementierung als grundsätzlich zweckdienlich beschrieben, wobei insbesondere eine hohe Stabilität (Robustheit und Kontrolle) beobachtet wird. Integrationen mit File-Imports werden als fehleranfällig beschrieben. Die fehlende Abstraktion bei datei- und datenbankbasierten Integrationen schränkt zudem die Flexibilität für Änderungen am Integrationsökosystem stark ein. Als am besten geeignet für ein operatives Data Warehouse wurde das Extrahieren der Daten über Webservices identifiziert.

Im Application Assessment werden für das operative Data Warehouse insbesondere Stärken bei Security and Compliance (Robustheit und Kontrolle) bescheinigt. Schwächen werden in Bezug auf „API-First“ (Latenz) und „Modular Architecture“ (Flexibilität) aufgezeigt, was sich mit der Beobachtung aus der Fallstudie deckt, in der Webservices als präferierte Lösung aufgezeigt werden.

Die Empfehlung des GAP-Modells für die Integrationsarchitektur für das operative Data Warehouse deckt sich mit den Beobachtungen aus der Fallstudie, sowie dem Ergebnis des Application Assessments. Demnach sollte eine Integration idealerweise nach Storage-Driven-Paradigma mit Webservices, oder Webhooks in einer Tree-Topologie implementiert werden. Die tatsächlichen Implementierungen erhalten im GAP-Modell eine gute Bewertung in Bezug auf Robustheit, Durchsatz und Kontrolle. Dem gegenüber steht eine schlechte Bewertung von Latenz und Integrität bei File-Imports, sowie eine schlechte Bewertung der Flexibilität der Integrationslandschaft bei direkten Datenbankzugriffen.

Fallstudie 1 zeigt somit, dass das GAP-Modell valide Ergebnisse liefert und zu korrekten Designempfehlungen führt.

6.5.2. Fallstudie 2: Anbindung von Satellitensystem an operative Kernsysteme

Für die in Abschnitt 6.2 beschriebene Fallstudie 2 zur Anbindung von Satelliten an ein operatives Kernsystem erfolgt eine separate Betrachtung pro angebundene Satelliten.

Satellit 1 wird als sehr robust beschrieben, wobei Anpassungen nur mit viel Aufwand umgesetzt werden können (Flexibilität). Zudem ist die Integration durch eingeschränkte Kontrolle und Integrität auf lesenden Zugriff beschränkt. Diese eingeschränkte Zukunftsfähigkeit und fehlende Flexibilität wird für Satellit 1 auch durch die Ergebnisse des Application Assessments bestätigt. Demnach entspricht die für Satellit 1 implementierte Integration nicht den Anforderungen und sollte für eine bessere Passung in Zukunft webbasiert umgesetzt werden.

In Zusammenhang mit Satellit 1 liefert das GAP-Modell plausible Ergebnisse. Die Anforderungen an Flexibilität und Kontrolle werden nicht erreicht, wobei die Integration als sehr robust bewertet wird, was jedoch in Anbetracht des Anforderungsprofils für die Bewertung der Integration nicht ins Gewicht fällt.

Satellit 2 erreicht durch das direkte Abfragen der aktuellen Daten eine sehr geringe Latenz. Als nachteilig wird beschrieben, dass Änderungen an der Datenstruktur mit hohem Aufwand verbunden sind. Dies schränkt die Flexibilität der Integrationslandschaft ein. Das Application Assessment zeigt, dass die Integration derzeit zweckdienlich implementiert ist (Latenz) und stabil betrieben wird (Robustheit). Aufgrund der Betriebsumgebung und abnehmender technischer Passung ist die betrachtete Integration jedoch als langfristig nicht passend markiert. Zudem deutet die schlechte Bewertung bei „API-First“ auf eingeschränkte Flexibilität durch die genutzte datenbankbasierte Integrationsmethode hin.

Die Ergebnisse des GAP-Modells für Satellit 2 bewerten die Integration in Bezug auf Latenz, Integrität und Skalierbarkeit als sehr gut. Gleichzeitig werden Schwächen bei der Kontrolle aufgezeigt und die Robustheit als schwach eingeschätzt. Die Ergebnisse des GAP-Modells stimmen auch in diesem Fall mit Bewertungen aus anderen Quellen überein.

Satellit 3 nutzt einen API-Wrapper, welcher sich anfangs durch Stabilitätsprobleme stark negativ auf die Robustheit der Integration ausgewirkt hat. Obwohl die Integration an sich zweckdienlich ist, wird nicht die erforderliche Kontrolle erreicht, um auch eine Integration mit schreibender Funktionalität zu erreichen. Die Ergebnisse des Application Assessments zeigen, dass die Anforderungen beinahe erfüllt sind, wobei jedoch Schwächen bei der Operational Excellence dokumentiert wurden, welche sich mit der schwachen Kontrolle und Robustheit in Verbindung bringen lassen.

Die Ergebnisse des GAP-Modells für Satellit 3 sind mit den Ergebnissen des Application Assessments plausibel in Einklang zu bringen, da auch hier das Anforderungsprofil beinahe in allen Punkten erfüllt wird. Eine sehr gute Latenz, Integrität und Skalierbarkeit steht mit starker Flexibilität und Kontrolle einer im Vergleich zu Satellit 1 etwas verringerten Robustheit entgegen. Obwohl die Kontrolle als stark bewertet ist, erfüllt die Integration nicht die benötigten Anforderungen, so dass auch dieser Archetyp einer Integrationsarchitektur für den betrachteten Anwendungsfall in Fallstudie 2 nicht vollumfänglich geeignet ist.

Die Integration von Satellit 4 mit dem operativen Kernsystem wird als einwandfrei beschrieben. Zwar ist die Robustheit aufgrund der zusätzlichen zentralen Komponente, sowie durch den genutzten API-Wrapper sehr eingeschränkt, dies steht jedoch einer sehr guten Kontrolle, ausgezeichneten Integrität und geringer Latenz gegenüber. Auch das Application Assessment zeichnet für Satellit 4 ein sehr positives Bild. Die Integration erfüllt alle Anforderungen, ist jedoch in Bezug auf Off-the-shelf und Operational Excellence schlecht bewertet, was sich mit der geringen Robustheit aufgrund von API-Wrapper und Integration-Hub deckt.

Die Ergebnisse für Satellit 4 sind eine weitere Bestätigung für die Validität des GAP-Modells. Die hier implementierte Integration nach Service-Driven-Paradigma mit Webservices über einen Integration-Hub entspricht genau den Empfehlungen des GAP-Modells. Kontrolle, Flexibilität, Integrität und Latenz erhalten in Übereinstimmung mit Application Assessment und dokumentierten Beobachtungen die Maximalbewertung. Im Kontrast dazu weist die Robustheit bei dieser Integrationsarchitektur den geringsten Wert auf.

6.5.3. Fallstudie 3: Kontinuierliche Synchronisierung von zweier operativer Kernsysteme

Fallstudie 3 ist in Abschnitt 6.4 beschrieben und untersucht die Integrationsarchitektur zur kontinuierlichen Synchronisierung zweier Kernsysteme im Parallelbetrieb. Hierbei wurden drei verschiedene Implementierungen untersucht. Da eine Integrationsarchitektur nur prototypisch implementiert wurde, liegen nur für zwei Implementierungen Ergebnisse des Application Assessments vor.

Die erste Implementierung der Integration in Fallstudie 3 ist mit File-Exports realisiert. Die tatsächliche implementierte Integration hatte anfänglich große Stabilitätsprobleme, welche insbesondere in Verbindung mit der schwachen Robustheit von API-Wrapper und Integration-Hub stehen. Abgefedert wurde dies durch das physische Kopieren der Daten entsprechend Storage-Driven-Paradigma. Die Schwäche in Bezug auf die Latenz der Integration hat zudem zunächst einen Parallelbetrieb verhindert und steht somit nicht mehr im Einklang mit den aktuellen Anforderungen.

Das Application Assessment charakterisiert die Implementierung mit File-Exports als flexibel und skalierbar (Modular Architecture und API-First) und attestiert eine ausreichende Integrität und Kontrolle (Security and Compliance). Als Schwäche wird hierbei eine große Lücke bei der Operational Excellence und der Zukunftsfähigkeit der technischen Umsetzung identifiziert, welche sich in den Zusammenhang mit geringer Robustheit und zu hoher Latenz bringen lassen.

Das GAP-Modell bewertet die Integration mit File-Exports als sehr stark beim Durchsatz, jedoch mit eklatanten Lücken bei Robustheit und Latenz. Dies deckt sich sowohl mit den oben beschriebenen Beobachtungen als auch den Ergebnissen des Application Assessments.

In der Fallstudie 3 ist eine Diskrepanz zwischen GAP-Modell und Application Assessment aufgefallen: Im Application Assessment wird die Kategorie „API-First“ gut bewertet, obwohl die Integration nach GAP-Modell als File-Export kategorisiert wurde und somit insbesondere Schwächen bei Latenz und Flexibilität zu erwarten sind. Davon abgesehen stehen die Ergebnisse des GAP-Modells jedoch in plausiblen Zusammenhang mit den Beobachtungen und Ergebnissen des Application Assessments.

Die zweite Implementierung in Fallstudie 3 nutzt Webservices und wurde als Weiterentwicklung der ersten Implementierung für den Parallelbetrieb beider Systeme eingeführt. Die Synchronisierung beider Systeme erfolgt ausreichend stabil, da durch die Storage-Driven-Integration eine starke Robustheit erreicht werden kann. Auch Latenz, Integrität und Kontrolle sind für den Parallelbetrieb ausreichend.

Das Application Assessment bewertet die betrachtete Integration mit Webservices grundlegend positiv. Es besteht jedoch Verbesserungspotential in den Kategorien „Modular Architecture“, sowie „Security and Compliance“. Diese Lücke kann in Verbindung mit der eingeschränkten Robustheit gesetzt werden.

Das GAP-Modell zeichnet ein mit den Beobachtungen und Application Assessment übereinstimmendes Bild. Bis auf die Dimension Robustheit sind alle Anforderungen erfüllt. Die Integrationsarchitektur erreicht bei Latenz, Integrität, Flexibilität und Kontrolle sogar Bewertungen, welche die Anforderungen übersteigen. Durch die Änderung von Hub-Topologie zu einem Integration-Tree würde die zentrale Komponente wegfallen. So könnte die geforderte sehr starke Robustheit erreicht werden, ohne dass die übrigen Punkte die Anforderungen unterschreiten.

Als dritte Integrationsarchitektur in Fallstudie 3 wird eine prototypische Implementierung nach Event-Driven-Paradigma betrachtet. Diese zeichnet sich durch eine äußerst geringe Latenz aus, indem Datenänderungen sofort in Echtzeit als Events übertragen werden. Der Prototyp wurde

jedoch nicht weiterverfolgt, da die Nachteile einer sehr geringen Robustheit und eingeschränkten Kontrolle die Vorteile einer Echtzeitintegration nicht aufwiegen konnten. Im Zusammenhang mit dem GAP-Modell ist auffällig, wie klar in diesem Fall modellbasierte Bewertung zur beobachteten Situation bei der tatsächlichen Implementierung abbildet. Bei einer Event-Driven-Integration mit Webservices in einem Integration-Hub sind im GAP-Modell alle Anforderungen bis auf Robustheit übererfüllt. Latenz, Integrität, Kontrolle und Flexibilität erreichen Höchstwerte, wohingegen Robustheit nur den Minimalwert erreicht. Dies zeigt den Wert des GAP-Modells bei der Wahl einer passenden Integrationsarchitektur, welche unter Anwendungen dieses modellbasierten Ansatzes innerhalb weniger Minuten erfolgen kann. So werden unnötige Fehlinvestitionen vermieden, da mögliche Schwächen von Integrationsarchitekturen bereits von Anfang an aufgezeigt werden.

6.5.4. Fazit

Zusammenfassend stellt der Autor fest, dass das GAP-Modell valide Aussagen zur Eignung von Integrationsarchitekturen liefert. Die Ergebnisse des GAP-Modells können für alle Fallstudien plausibel in Zusammenhang mit den Ergebnissen des Application Assessments, sowie der Beobachtungen bei der tatsächlichen Implementierung gebracht werden. Die im geringen Maße identifizierten Diskrepanzen ergeben sich aus den unterschiedlichen Betrachtungsweisen und Bezugssystemen der verglichenen Bewertungsmodelle und sind für die Güte der Architekturbewertungen und -empfehlungen des GAP-Modells unerheblich. Somit ist das GAP-Modell ein geeignetes Werkzeug zur Unterstützung bei der Modernisierung historisch gewachsener Systemlandschaften und insbesondere für die Wahl passender Integrationsarchitekturen sinnvoll.

7. Bewertung und Ausblick

7.1. Bewertung der Forschungsergebnisse

Die in Kapitel 6 beschriebenen Ergebnisse der Validierung des GAP-Modells erlauben eine positive Bewertung der Forschungsergebnisse. In allen betrachteten Fallstudien lieferte das GAP-Modell korrekte Einschätzungen zu Stärken und Schwächen von Integrationsarchitekturen. Das GAP-Modell kann sowohl für die Identifikation von Handlungsbedarf in bestehenden Architekturen als auch für neue Designentscheidungen genutzt werden. Die These „*Integrationsarchitekturen lassen sich modellgetrieben optimieren*“ lässt sich also klar mit Ja beantworten.

Der Autor schlägt mit dem GAP-Modell eine schlüssige Methode zur Verbesserung organisatorischer Leistungsfähigkeit bei der digitalen Transformation unter dem Aspekt des Fachkräftemangels und der massenhaften API-fizierung vor. Großer Vorteil des GAP-Modells ist neben der klaren Strukturierung von Architekturen die einfache Handhabung, welche einen schnellen und ressourcensparenden Einsatz und Integration in bestehende Strukturen ermöglicht.

7.1.1. Reflektion der Forschungsfragen

Der Autor beantwortet in Kapitel 2 zunächst die Forschungsfragen 1.1, 1.2 und 1.3 zur modellbasierten Optimierung der Systemintegration. Etablierte Ansätze zur Modellierung und Gestaltung von Systemlandschaften werden dabei auf die Problemstellung dieser Arbeit transferiert.

Analytisches Kernstück dieser Arbeit ist die Erarbeitung einer Taxonomie zur Kategorisierung von Integrationsarchitekturen durch Segmentierung anhand bestimmender Merkmale. Dieser Ansatz Architekturen in Artefakte zu zerlegen, diese auf verschiedenen Granularitätsebenen zu betrachten und vertikal in Klassen zu segmentieren ist Voraussetzung für die vergleichende Bewertung dieser Architekturausprägungen. Die darauf aufbauende Archetypisierung ermöglicht eine klare Strukturierung aller Integrationsarchitekturen im Schema des GAP-Modells. Durch die Einbettung in das TOGAF Architekturrahmenwerk kann diese Methode standardisiert und massenhaft eingesetzt werden. Die Möglichkeiten zur globalen Kalibrierung erlauben es Nutzern zudem das Modell für eine bestimmte Einsatzumgebung einzustellen und an den jeweiligen Kontext anzupassen. Die Forschungsfrage 2, nach welcher Systematik Integrationsarchitekturen unterschieden werden können und welche Klassifizierung sich daraus ergeben wird somit klar strukturiert, erfolgreich und innovativ beantwortet.

Das in Kapitel 4 ausgearbeitete GAP-Modell verbindet etablierte Ansätze mit der Idee, dass über eine Systematisierung und Archetypisierung von Integrationsarchitekturen automatisierbar bestehende Systemintegrationen bewertet und passende Integrationsarchitekturen identifiziert werden können. Gemeinsam mit der Parametrisierung und Validierung des GAP-Modells in Kapitel 5 und Kapitel 6 beantwortet der Autor damit die finale Forschungsfrage 3 zur Bewertung von Integrationsarchitekturen.

Dabei vergleicht der Autor auf Grundlage der erarbeiteten Taxonomie verschiedene Architektursegmente von Integrationsarchitekturen und prüft mit diesen Parametern die Plausibilität der Ergebnisse des GAP-Modells. Die Richtigkeit des GAP-Modells wird dabei anhand von Fallstudien abschließend empirisch validiert. Diese Fallstudien wurden anhand realer Anwendungsfälle durchgeführt. Der Autor kombiniert also umfassende theoretischen und analytischen Überlegungen mit einem hohen Praxisbezug, so dass die Ergebnisse dieser Arbeit eine hohe wissenschaftliche Güte haben und gleichzeitig direkt in der Praxis der Wirtschaftsinformatik anwendbar sind.

7.1.2. Potential zur Optimierung und weiteren Vertiefung

Optimierungspotential identifiziert der Autor vor allem bei der weiteren Ausarbeitung der Architekturbewertungen. Der Autor fokussiert im Rahmen dieser Arbeit zunächst auf die Validierung des Bewertungsmodells und verzichtet daher auf eine tieferen Detailgrad bei der analytischen Untersuchung der Stärken und Schwächen der jeweiligen Architekturklassen. Hier besteht Potential zur Weiterentwicklung und wissenschaftlichen Vertiefung, welche die initiale Parametrisierung des GAP-Modells möglicherweise verbessern würde. So könnten diese Bewertung anhand empirischer Studien feinjustiert, oder im Rahmen selbstlernender Algorithmen im tatsächlichen Betrieb kontinuierlich verbessert werden. Auch eine höhere Auflösung der Bewertungen durch eine feinere Gliederung der Skala würde die Ergebnisse des GAP-Modells eventuell verbessern. Nichtsdestotrotz zeigen die Forschungsergebnisse, dass bereits mit der für die erste Validierung gewählten Auflösung und Parametrisierung sehr klare und plausible Architekturempfehlungen erreicht werden können.

Eine weitere Verbesserungsmöglichkeit besteht in der vereinfachten Anwendung des Modells. Durch die Implementierung eines elaborierten Expertensystems könnten beispielsweise Berechnungen weiter automatisiert und verbessert werden. Der Autor nutzt derzeit zur Berechnung der Bewertungen ein einfaches Kalkulationsprogramm. Mit der Entwicklung eines Expertensystems, oder der Einbindung in ein bestehendes Enterprise-Architektur-Werkzeug könnten weitere Schritte automatisiert, abstrahiert und mit unterstützenden Informationen begleitet werden, so dass die Anwendungsschwelle in Bezug auf die benötigte Expertise weiter gesenkt werden könnte. Insbesondere bei der stringenten Ermittlung der Anforderungsprofile, sowie der Einarbeitung von betrieblichen, technischen und organisatorischen Rahmenbedingungen könnte dies im Sinne einer Architektur-Governance im Enterprise-Kontext sinnvoll sein.

Eine entscheidende Voraussetzung für die richtige Anwendung des GAP-Modells ist die korrekte Kategorisierung bestehender Integrationsarchitekturen und Interpretation der Bewertungsergebnisse. Die Kategorisierung im GAP-Modell ist zwar klar und einfach, erfordert jedoch bei der Bewertung bestehender Strukturen nach wie vor ein tiefes Verständnis für Integrationsarchitekturen. Eine signifikante Verbesserung könnte daher zum Beispiel mit der automatisierten Identifikation von Architekturmustern aus Modelldarstellungen erfolgen. Fallstudie 3 hat gezeigt, dass es hier durchaus zu Fehlern und Ungereimtheiten kommen kann, die eine Interpretation der Bewertungsergebnisse stark erschweren. Voraussetzung hierfür ist eine maschinenlesbare Modellierung der betrachteten Systemstrukturen (beispielsweise mit UML) die eine automatisierte Musteridentifikation erlaubt.

Abschließend ist auch die Erweiterung des GAP-Modells auf zusätzliche tiefere Granularitätsebenen denkbar und würde noch differenziertere Aussagen ermöglichen. So könnte beispielsweise neben der Methode („Was“) auch noch das „Wie“ aufgezeigt werden, indem zum Beispiel Best-Practices und konstruktive Designmuster auf APIs verschiedener Methoden Anwendung finden. Somit wäre nicht nur eine Messung und Optimierung der Architektur, sondern auch die Steuerung der jeweiligen Implementierungen möglich.

Die Anwendung des GAP-Modells erfordert noch ausgeprägtes Fachwissen in Bezug auf die IT-Architektur. Es kann also insbesondere als Expertensystem gesehen werden, welches den Bedarf für gute IT-Architekten nicht verringert, deren Effizienz jedoch durch ein einfaches und automatisierbares Bewertungssystem deutlich steigert. Ein weiterer Schritt wäre eine weitere Vereinfachung der Anwendung und die Integration von Erklärungen und Beispielimplementierungen als Ergebnis der Bewertung im GAP-Modell. Eine solche Demokratisierung ist bereits in Teilen der Softwareentwicklung zu beobachten und könnte auch auf die IT-Architektur übertragen werden.

7.2. Ausblick

In folgendem werden ausgewählte Überlegungen zum Optimierungspotential aufgegriffen und näher beleuchtet.

7.2.1. KI-gestützte Automatisierung des GAP-Modells

Die aktuelle Beschleunigung der KI-Transformation eröffnet bahnbrechende Möglichkeiten zur Digitalisierung und Automatisierung betrieblicher Abläufe und hebt potentiale zur Skalierung von Ergebnissen. [82, Seite 11]

In Bezug auf die Digitalisierung mit APIs eröffnen sich mit künstlicher Intelligenz Möglichkeiten zur Automatisierung des GAP-Modells. Wie oben beschrieben, ist eine Kernvoraussetzung für die Anwendung des GAP-Modells die korrekte Identifikation von Mustern von Designartefakten, so dass den zu bewertenden Integrationsarchitekturen ein passender Archetyp zugeordnet werden kann. Im Idealfall liegen standardisierte Modelle für alle relevanten Komponenten der Systemlandschaft in einem zentralen Repository vor, so dass diese einfach verfügbar und auswertbar sind.

In der Praxis ist die Dokumentation über den Systemlebenszyklus jedoch nicht stringent, so dass sich relevante Informationen in einer Vielzahl von Dokumenten und grafischen Abbildungen finden und bestimmte Implementierungsmuster nur über den vorliegenden Quellcode oder Projektdateien verfügbar gemacht werden können. Diese Informationen zusammenzutragen, zu strukturieren und auszuwerten ist Kernaufgabe eines IT-Architekten bei der Modernisierung historisch gewachsener Systemlandschaften.

Diese Aufgabe könnte mit KI deutlich effizienter umgesetzt werden, indem beispielsweise alle relevanten Dokumente automatisch zusammengetragen und ausgewertet werden. Eine KI könnte auf Grundlage der Dokumentation, sowie insbesondere durch Auswertung von Code und Kommunikationsmustern die tatsächlich implementierten Muster erkennen und IT-Architekten damit enorm entlasten. Zudem könnten diese Informationen dann beispielsweise in UML überführt werden, so dass ein wirkliches Architektur-Repository entsteht. IT-Architekten könnten sich dann auf die Optimierung der identifizierten Strukturen, sowie den Entwurf moderner Integrationsarchitekturen konzentrieren.

Basierend auf den Empfehlungen des GAP-Modells könnte eine Künstliche Intelligenz Implementierungsmuster vorschlagen und den entsprechenden Code unter Anleitung eines IT-Architekten generieren, so dass eine massenhafte API-fizierung mit sehr beschränkten personellen Ressourcen ermöglicht wird. Der Autor hat diesen Ansatz bereits prototypisch untersucht und unter Angabe von Architekturmustern stichprobenartig Code über openAI mit GPT-3.5 generiert (siehe Anhang 15, Seiten 170ff).

7.2.2. Spezifikationsbasierte Automatisierung

Im Zusammenhang mit der KI-gestützten Automatisierung von Arbeitsabläufen bei der Gestaltung und Implementierung von Systemintegrationen steht insbesondere die spezifikationsbasierte Automatisierung als etablierter Ansatz zur Optimierung im Softwareengineering. Im Kontext von REST-APIs sind Standardspezifikationsformate, wie OAS oder RAML bereits etablierte Werkzeuge, um basierend auf einer Spezifikation gezielt Code zu generieren, APIs zu testen, Zugriffe zu steuern und die Kommunikation zwischen Akteuren im Konnektivitätsökosystem zu überwachen.

Mögliche Skaleneffekte durch spezifikationsbasierte Automatisierung wurden vom Autor bereits in Bezug auf Sicherheits- und Compliance-Management im Lebenszyklus von Web-APIs erörtert. [43] Konzepte wie Contract-Driven-Development und das Modell des ganzheitlichen API-Managements sind auch im weiteren Kontext der Digitalisierung mit APIs anzuwenden. Anhand von standardisierter Spezifikation können Standardkomponenten und -lösungen implementiert und kontinuierliche wiederverwendet werden. Zudem können auch dedizierte Integrationsplattformen mit vorliegenden Spezifikationen gefüllt werden, so dass einfach und unkompliziert Wrapper und Mediatoren implementiert bzw. konfiguriert werden können.

Somit eröffnen inhaltliche und technische Standards und Normen zum einen Effizienzpotentiale bei Entwicklung und Betrieb von Systemintegrationen, senken Komplexität und Kosten und erhöhen die Sicherheit und Compliance im Konnektivitätsökosystem.

Am Beispiel von OpenAPI ist erkennbar, wie sich spezifikationsbasierte Automatisierung für einen Teilbereich der Systemintegration bereits stark etabliert hat: 2015 nutzen nur 25% der im SmartBear „State of API“ Report befragten den OpenAPI Standard. Innerhalb von drei Jahren hat sich diese Zahl verdreifacht. [38] Bereits zwei Jahre später hat sich OpenAPI als de-facto Industriestandard etabliert und wird dementsprechend seit 2020 standardmäßig in voller Breite zur Spezifikation von webbasierten Schnittstellen genutzt. [83]

7.2.3. Demokratisierung der Softwareentwicklung

Die Demokratisierung der Softwareentwicklung bietet Möglichkeiten zur Beschleunigung der digitalen Transformation. Die große Nachfrage nach individuellen digitalen Lösungen trifft auf das Problem sehr begrenzter Kapazitäten in der Softwareentwicklung. Ein Lösungsansatz bieten hierbei Low-Code/No-Code Plattformen, bei denen Nutzer aus ihrem Betrieblichen Kontext heraus Applikationen entwickeln, ohne dass sie die hierfür benötigten Fähigkeiten des klassischen Softwareengineerings verfügen. [84, Seite 3]

Die Demokratisierung der Softwareentwicklung steht jedoch noch am Anfang. Zwar nutzen viele Unternehmen bereits Low-Code-Ansätze, diese konnten sich jedoch noch nicht als echte Alternative zur klassischen Softwareentwicklung durchsetzen. Ein Hauptgrund hierfür ist, dass angebotene Plattformen meistens nur eine begrenzte Anwendungsbreite anbieten und auf spezielle Anwendungsfälle beschränkt sind [84, Seite 12]

Im Kontext der Digitalisierung mit APIs bedeutet die Demokratisierung der Softwareentwicklung vor allem, dass Entscheidungsprozesse für das Design von Systemintegrationen in Zukunft verstärkt fachfremde Teilnehmer einbeziehen müssen. Wenn Fähigkeiten des klassischen Softwareengineerings nicht mehr zwingend vorhanden sind, dann muss der Entwicklungsprozess von Schnittstellen und Interaktionen im Konnektivitätsökosystem so robust und einfach wie möglich gestaltet sein. Es besteht die Gefahr, dass durch die fehlende Expertise die Problematik der „Accidental Architecture“ weiter an Gewicht gewinnt.

In diesem Kontext bietet das GAP-Modell einen einfach zu nutzenden Ansatz zur Herbeiführung passender Designentscheidungen für Systemintegrationen. Wenn dieses Modell in den Entwicklungsprozess eingebettet wird und in einfach von allen Beteiligten genutzt werden kann, dann können unpassende Designentscheidungen identifiziert werden und es wird aufwandsarm auf eine passende Integrationsarchitektur hingewirkt. Über KI-gestützte und spezifikationsbasierte Automatisierung können in diesem Zusammenhang auch Implementierungen vorgeschlagen und entsprechend den Anforderungen (semi-)automatisiert umgesetzt werden.

7.2.4. Flexibilisierung mit domänenspezifischen Datenmodellen

Abschließend soll auf die Möglichkeiten zur Flexibilisierung von Systemintegrationen mit domänenspezifischen Datenmodellen eingegangen werden.

Sowohl allgemeine, als auch Domänenspezifische Datenmodelle sind ein etablierte Standardisierungsmöglichkeit zur Erreichung von Interoperabilität. Um die Interoperabilität und Orchestrierung verschiedener APIs zu erleichtern, sollten zunächst allgemeine Standards gelten. Beispiele hierfür sind die Formatierung von Datum und Uhrzeit nach ISO-Norm, GeoCodes, Namen, Adressen, etc. Zusätzlich sollte sich die Bezeichnung von Objekten und Attributen an einem gemeinsamen fachlichen Standard orientieren. Beispiele für domänenspezifische Standardformate und -modelle sind:

- Finanzen – Europäische Zentralbank (EZB)
 - PSD2: Payment Service Directive als zentrale Zahlungsdoemsrichtlinie mit standardisierten Schnittstellenanforderungen [85]
- Klinische Forschung - Clinical Data Interchange Standards Consortium (CDISC)
 - ADaM: Analysis Data Model als regulatorischer Standard für die Analyse von Daten in klinischen Studien [86]
 - SDTM: Study Data Tabulations Model als regulatorischer Standard für die Ablage und Übertragung von Daten in klinischen Studien [87]
 - BRIDGE: Biomedical Research Integrated Domain Group als vielschichtiges Domänenstandardmodell für klinische Studien [88]
- Bauwesen – buildingSMART e.V.
 - IFC: Industry Foundation Classes als offener Standard für die ganzheitliche Modellierung Bauvorhaben / Building Information Modelling (BIM) [89]

Der Aspekt von Standardisierung zur Flexibilisierung kann im GAP-Modell insbesondere die Ebene der Integrationsmethode erweitern. In der Bewertung der der Integrationsmethoden wurde bereits berücksichtigt, dass eventuelle Wrapper und Mediatoren der Robustheit der Integrationen nicht zuträglich sind. Positive Auswirkungen werden in diesem Zusammenhang derzeit im GAP-Modell jedoch nicht vorgesehen. Wird beispielsweise ein API-Orchestrierung genutzt, um ein Domänenstandardmodell für die bestehenden Schnittstellen zu implementieren, so schwächt dies zwar die Robustheit, hat aber gleichzeitig auch einen positiven Einfluss auf die Flexibilität und Skalierbarkeit der Integration. Ein weiteres Beispiel wäre die Implementierung eines Wrappers, der Dateien nun in einem Standardformat zur Verfügung stellt und somit die Robustheit und Integrität dateibasierter Integrationen deutlich steigert. In diesem Fall ist muss die Betrachtung von Wrapper/Mediator im GAP-Modell entweder verfeinert werden, oder die Bewertung im jeweiligen Anwendungsfall manuell angepasst werden.

8. Quellenverzeichnis

- [1] Food and Drug Administration: General Principles of Software Validation. Guidance for Industry and FDA Staff, 2002. <https://www.fda.gov/regulatory-information/search-fda-guidance-documents/general-principles-software-validation>, abgerufen am: 18.02.2023
- [2] Institut der Deutschen Wirtschaft: Lagebild Digitalisierung - Was ist Digitalisierung?, 2022. <https://www.de.digital/DIGITAL/Navigation/DE/Lagebild/Was-ist-Digitalisierung/was-ist-digitalisierung.html>, abgerufen am: 24.02.2023
- [3] Sensler, C.: APIs: Turbo der Digitalisierung. APIfication als Enabler von plattformbasierten Geschäftsmodellen, 2018. <https://entwickler.de/online/web/api-digitalisierung-579827381.html>. aktualisiert am: 14.06.2018, abgerufen am: 12.10.2019
- [4] Institut der Deutschen Wirtschaft: Digitalisierungsindex 2022, Berlin 2022. <https://www.de.digital/DIGITAL/Redaktion/DE/Digitalisierungsindex/Publikationen/publikation-digitalisierungsindex-2022-langfassung.pdf>. aktualisiert am: Januar 2023, abgerufen am: 24.02.2023
- [5] Dedić, N.: FEAMI: A Methodology to Include and to Integrate Enterprise Architecture Processes Into Existing Organizational Processes. IEEE Engineering Management Review 48 (2020) 4 SS. 160–166. <https://ieeexplore.ieee.org/document/9234599/>
- [6] The Open Group: TOGAF Standard - Introduction and Core Concepts. Reading, United Kingdom 2022, ISBN: 1-947754-90-4
- [7] Georgi, A.: Theorie und Einsatz von Verbindungseinrichtungen in parallelen Rechnersystemen - Klassifizierung von Verbindungsnetzwerken. Vorlesungsunterlagen, 2012. https://tu-dresden.de/zih/ressourcen/dateien/lehre/ss2012/tevpr_content/V2_Klassifizierung_4_on_1.pdf?lang=en. aktualisiert am: Sommersemester 2012, abgerufen am: 03.10.2023
- [8] Baun, C.: Computer Networks. Bilingual Edition: English - German / Zweisprachige Ausgabe: Englisch - Deutsch. Wiesbaden: Springer Gabler. in Springer Fachmedien Wiesbaden GmbH 2019, ISBN: 978-3-658-26356-0
- [9] The Open Group: TOGAF Standard - Applying the ADM. Reading, United Kingdom 2022, ISBN: 1-947754-90-4
- [10] Leser, U., Busse, S., Kutsche, R.-D. u. Weber, H.: Federated Information Systems. Concepts, Terminology and Architectures, Berlin 1999. https://www.researchgate.net/publication/237380766_Concepts_Terminology_and_Architectures. aktualisiert am: January 1998, abgerufen am: 03.10.2023
- [11] Reese, J.: Wirtschaftsinformatik. Eine Einführung. Wiesbaden: Gabler Verlag 1990, ISBN: 978-3-322-86440-6
- [12] Leser, U.: Informationsintegration: Mediator-basierte Integrationsysteme. Vorlesungsunterlagen, Berlin 2016. aktualisiert am: Sommersemester 2016, abgerufen am: 03.10.2023
- [13] ISO/IEC 27000:2018-02. *Information technology — Security techniques — Information security management systems*. https://standards.iso.org/ittf/PubliclyAvailableStandards/co73906_ISO_IEC_27000_2018_E.zip, abgerufen am: 03.10.2023
- [14] Britze, N., Schulze, A., Fenge, K., Woltering, M., Gross, M., Menge, F., Mucke, A., Ensinger, A., Keller, H., Oldenburg, L., Wederhake, A., Backers, S., Braunschweig, M.-C. von, Wankel, A., Rapp, C., Sturm, T., Hoffmann, G., Laux, M., Schröder, T., Zibell, M., Hoffmann, D.,

- Streif, M. u. Halstenbach, V.: Leitfaden zum Reifegradmodell Digitale Geschäftsprozesse, Berlin 2020. aktualisiert am: 2020, abgerufen am: 14.01.2021
- [15] Britze, N. u. Danneberg, M.: Leitfaden zum Reifegradmodell Digitale Geschäftsprozesse 2.0, Berlin 2022. <https://www.bitkom.org/sites/main/files/2022-12/221202LFreifegradmodellDigitale-Geschäftsprozesse-20.pdf>. aktualisiert am: December 2022, abgerufen am: 24.02.2023
- [16] Newman, D.: Top Five Digital Transformation Trends In Health Care, 2017. <https://www.forbes.com/sites/danielnewman/2017/03/07/top-five-digital-transformation-trends-in-healthcare>. aktualisiert am: 07.03.2017, abgerufen am: 03.10.2023
- [17] Paul, S. M., Mytelka, D. S., Dunwiddie, C. T., Persinger, C. C., Munos, B. H., Lindborg, S. R. u. Schacht, A. L.: How to improve R&D productivity. The pharmaceutical industry's grand challenge. *Nature Reviews Drug Discovery* 9 (2010)
- [18] Gupta, S. K.: Drug discovery and clinical research. New Delhi: Jaypee Brothers Medical Publishers 2011, ISBN: 978-93-5025-272-7
- [19] Kayyali, B., Knott, D. u. van Kuiken, S.: The big-data revolution in US health care: Accelerating value and innovation, 2013. http://healthcare.mckinsey.com/sites/default/files/The_big-data_revolution_in_US_health_care_Accelerating_value_and_innovation%5B1%5D.pdf, abgerufen am: 16.03.2017
- [20] Li, G., Jenkins, A., Arlington, S. u. Chew, P.: Performance-Based Site Selection Reduces Costs and Shortens Enrollment Time. *Pharmaceutical Executive* (2018). <https://www.pharmexec.com/view/clinical-trials-dont-have-cost-too-much-or-take-too-long-1>, abgerufen am: 03.10.2023
- [21] Chappell, D. A.: Enterprise service bus. Theory in practice. Beijing, Farnham: O'Reilly 2004, ISBN: 978-0-596-00675-4
- [22] Hartenstein, S., Nadobny, K., Schmidt, S. u. Schmietendorf, A.: Tagungsbericht "Evaluation of Service-APIs – ESAPI 2020". APIs als Klebstoff einer allumfassenden Digitalisierung, 2020. https://www.researchgate.net/publication/348742030_Workshop_Evaluation_of_Service-APIs_ESAPI_2020_Motto_APIs_als_Klebstoff_einer_allumfassenden_Digitalisierung_Detaillierter_Bericht. aktualisiert am: April 2021, abgerufen am: 03.10.2023
- [23] Kempermann, Hanno, Millack u. Agnes: Digitale Regionen in Deutschland: Ergebnisse des IW-Regionalrankings 2018. *IW-Trends - Vierteljahresschrift zur empirischen Wirtschaftsforschung* 45 (2018) 1 SS. 49–66. <https://www.econstor.eu/handle/10419/181923>, abgerufen am: 03.10.2023
- [24] Ematinger, R.: Von der Industrie 4.0 zum Geschäftsmodell 4.0. Chancen der digitalen Transformation. Essentials. Wiesbaden: Springer Gabler 2018, ISBN: 978-3-658-19474-1
- [25] Rajabalinejad, M., van Dongen, L. u. Ramtahaling, M.: Systems integration theory and fundamentals. *Safety and Reliability* 39 (2020) 1 SS. 83–113
- [26] Hasselbring, W.: Information System Integration. *Commun. ACM* 43 (2000) 6 SS. 32–38
- [27] Lackes, R. u. Siepermann, M.: Medienbruch. Definition im Gabler Wirtschaftslexikon, 2018. <https://wirtschaftslexikon.gabler.de/definition/medienbruch-51830/version-274981>. aktualisiert am: 19.02.2018, abgerufen am: 27.02.2023

- [28] Biswas, J., Lazar, A. A., Huard, J.-F., Lim, K., Mahjoub, S., Pau, L.-F., Suzuki, M., Torstensson, S., Wang, W. u. Weinstein, S.: The IEEE P1520 standards initiative for programmable network interfaces. *IEEE Communications Magazine* 36 (1998) 10 SS. 64–70
- [29] Nadobny, K. (Hrsg.): API-fizierung von Legacy-Systemen als zentraler Aspekt einer zeitgemäßen Systemintegrationsstrategie. Beitrag zum 2. Workshop Evaluation of Service-APIs, 08. November 2018, München. *Berliner Schriften zu modernen Integrationsarchitekturen*, Bd. 19. Aachen: Shaker 2018, ISBN: 9783844062540
- [30] Bitkom: Open Data – Neue Konzepte erfolgreich umsetzen, 2017. <https://www.bitkom.org/sites/default/files/file/import/171103-Open-Data.pdf>. aktualisiert am: 03.11.2017, abgerufen am: 03.10.2023
- [31] NIST Big Data Interoperability Framework, NIST Special Publication 1500-6r1, Chang, W., Boyd, D. u. NIST, N., Gaithersburg, MD 2018
- [32] Markgraf, D.: Prosumer - Ausführliche Definition im Online-Lexikon, 2018. <https://wirtschaftslexikon.gabler.de/definition/prosumer-42486/version-265834>. aktualisiert am: 16.02.2018, abgerufen am: 03.10.2023
- [33] Perry, D. E. u. Wolf, A. L.: Foundations for the Study of Software Architecture. *SOFTWARE ENGINEERING NOTES* 17 (1992) 4 SS. 40–52
- [34] IEEE: What is an API. https://developer.ieee.org/What_is_an_API#:~:text=An%20API%20%28Application%20Programming%20Interface%29%20is%20a%20software,An%20API%20specifies%3A%20What%20information%20can%20be%20requested, abgerufen am: 27.02.2023
- [35] Nitze, A.: APIs als Schlüssel zur digitalen Transformation 20 (2019) SS. 19–24
- [36] Steve Yegge: Stevey's Google Platforms Rant - Jeff Bezos Mandate, 2011. <https://gist.github.com/chitchcock/1281611>. aktualisiert am: 12.10.2011, abgerufen am: 03.10.2023
- [37] Nadobny, K. u. Schmietendorf, A.: Deutschsprachige Umfrage zur API Sicherheit. *Berliner Forum Digitalisierung. ESAPI* 21 (2019) SS. 75–84
- [38] SmartBear: The State of API. 2019
- [39] Nadobny, K.: API-fizierung von Legacy-Systemen im Kontext agiler Applikationsentwicklung. 3. Workshop Evaluation of Service-APIs, 07. November 2019, Dresden. *ESAPI* 21 (2019) SS. 53–66
- [40] Nadobny, K.: API-fizierung von Altanwendungen 20 (2019) SS. 49–56
- [41] Ried, S.: 4 große IT-Trends bestimmen 2019, 2019. <https://www.cio.de/a/4-grosse-it-trends-bestimmen-2019,3592542>. aktualisiert am: 18.07.2019, abgerufen am: 27.02.2023
- [42] Wood, C.: The Core Principles of API Management, 2016. <https://nordicapis.com/core-principles-api-management/>. aktualisiert am: 08.03.2016, abgerufen am: 03.10.2023
- [43] Schmietendorf, A., Schmidt, S., Nadobny, K. u. Hartenstein, S.: Sicherheits- und Compliance-Management im Lebenszyklus von Web-APIs. Ergebnisse eines Forschungsprojektes an der HWR Berlin/ Otto-von-Guericke-Universität Magdeburg. Berlin: Logos Berlin 2020, ISBN: 978-3-8325-5086-8
- [44] Chiu, D.: Where API management is headed in 2017. With full lifecycle capabilities, APIs mature to have a leading role in enterprise digital transformation., CA technologies 2017. <https://www.linkedin.com/pulse/where-api-management-headed-2017-david-chiu/>. aktualisiert am: 23.01.2017, abgerufen am: 03.10.2023

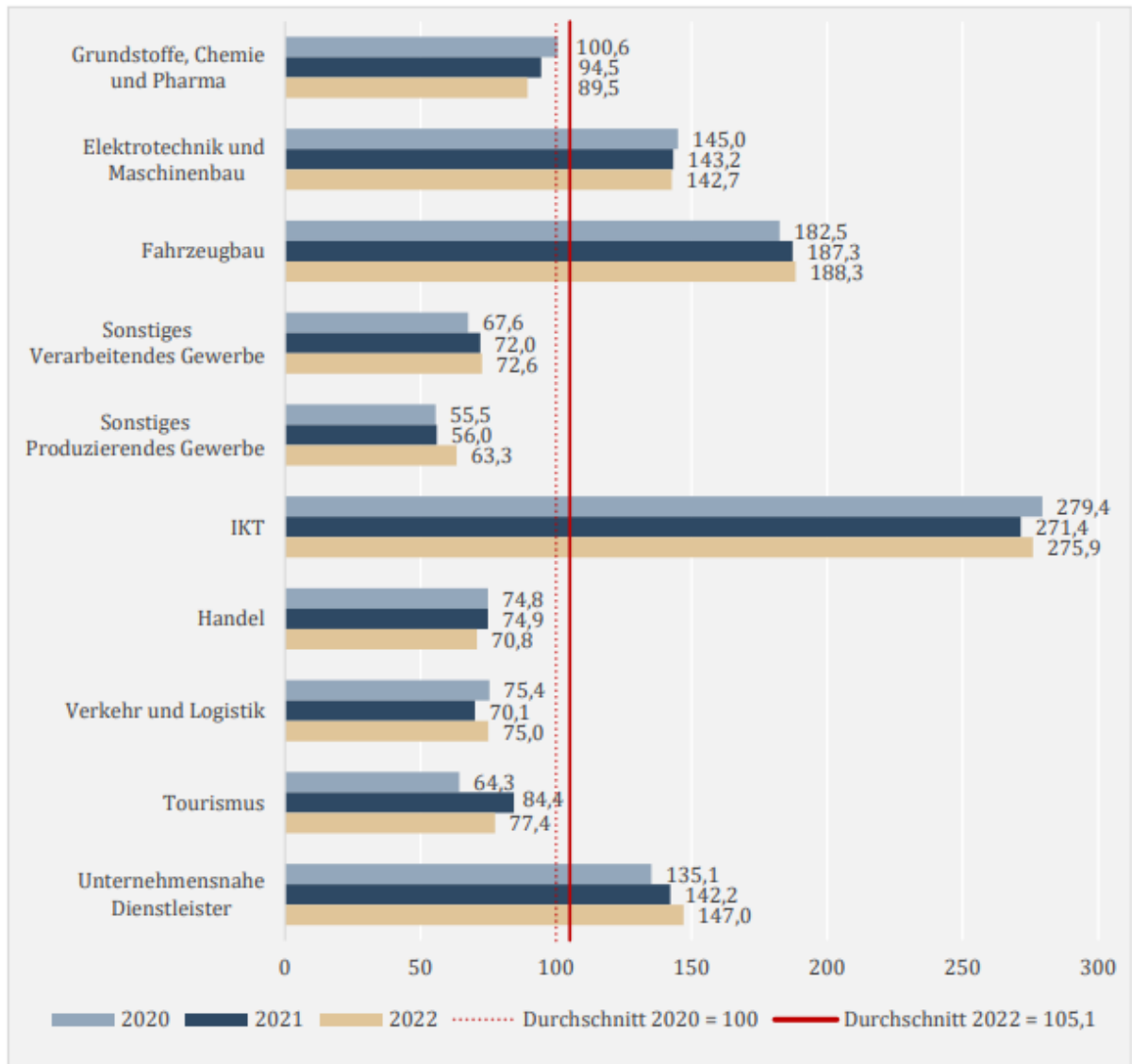
- [45] Pillai, S., Iijima, K., O'Neill, M., Santoro, J., Dumas, P. u. Jain, A.: Gartner Magic Quadrant for Full Life Cycle API Management, 2022. <https://www.gartner.com/en/documents/4021131>. aktualisiert am: 14.11.2022, abgerufen am: 03.10.2023
- [46] Nadobny, K. u. Schmietendorf, A.: ESAPI 2020. 4. Workshop Evaluation of Service-APIs. Berliner Schriften zu modernen Integrationsarchitekturen, Bd. 24. Düren: Shaker 2020, ISBN: 978-3-8440-7515-1
- [47] The Open Group: The TOGAF Standard, 10th Edition Translation Glossary. Reading, United Kingdom 2023, ISBN: 1-957866-27-7
- [48] The Open Group: TOGAF Standard - Architecture Content. Reading, United Kingdom 2022, ISBN: 1-947754-90-4
- [49] OMG: Unified Modeling Language Specification, 2017. <https://www.omg.org/spec/UML/2.5.1/PDF>. aktualisiert am: Dezember 2017, abgerufen am: 03.10.2023
- [50] Diestel, R.: Graphentheorie. Heidelberg: Springer 2012, ISBN: 978-3-642-14911-5
- [51] The Open Group: TOGAF Standard - Architecture Development Method. Reading, United Kingdom 2022, ISBN: 1-947754-90-4
- [52] Naeem, T.: Alles, was Sie über die Datenpipeline-Architektur wissen müssen, 2020. <https://www.astera.com/de/type/blog/data-pipeline-architecture/>. aktualisiert am: 22.02.2020, abgerufen am: 10.02.2023
- [53] Haas, Miller, Niswonger, Tork Roth, Schwarz u. Wimmers: Transforming Heterogeneous Data with Database Middleware: Beyond Integration. Bulletin of the Technical Committee on Data Engineering 22 (1999) 1 SS. 31–36
- [54] Dunkel, J.: Systemarchitekturen für verteilte Anwendungen. München: Hanser 2008, ISBN: 978-3-446-41745-8
- [55] Brown, P., Estefan, J., Laskey, K., McCabe, F. G. u. Thornton, D.: Reference Architecture Foundation for Service Oriented Architecture, 2012. <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra.pdf>. aktualisiert am: 04.12.2012, abgerufen am: 10.02.2023
- [56] MDN contributors: An overview of HTTP, 2020. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>. aktualisiert am: 15.05.2023, abgerufen am: 03.10.2023
- [57] Buckel, T.: Zum Potential von Event-Driven Architecture für komplexe Unternehmensnetzwerke. Multikonferenz Wirtschaftsinformatik 2012 : Tagungsband der MKWI 2012. (2012). <https://nbn-resolving.org/urn:nbn:de:gbv:084-13010815292>, abgerufen am: 03.10.2023
- [58] Erber, M.: IoT Data Streaming. Warum MQTT und Kafka eine exzellente Kombination sind _ Informatik Aktuell, 2021. <https://www.informatik-aktuell.de/betrieb/netzwerke/iot-data-streaming-warum-mqtt-und-kafka-eine-exzellente-kombination-sind.html>. aktualisiert am: 12.01.2021, abgerufen am: 10.02.2023
- [59] Amazon Web Services, Inc. or its affiliates: Building event-driven architectures on AWS, 2023. <https://d1.awsstatic.com/psc-digital/2023/gc-300/build-eda-on-aws/Build-EDA-AWS-eBook.pdf>, abgerufen am: 03.10.2023
- [60] Amazon Web Services, Inc. or its affiliates: What is a Message Queue?, 2019. <https://aws.amazon.com/message-queue/>, abgerufen am: 03.10.2023
- [61] 44302:1987-02. Informationsverarbeitung; Datenübertragung, Datenübermittlung

- [62] Fan, E., Agrawal, S., Cocchi, J., Rajagt, V. u. Dolan, L.: Azure Logic Apps. Übersicht über integrierte Connectors, 2022. <https://learn.microsoft.com/de-de/azure/connectors/built-in>. aktualisiert am: 06.12.2022, abgerufen am: 05.03.2023
- [63] West, R., Roth, J., surgos, V., Ray, M., Sherer, T., (None), M., Popovic, J., Guyer, C. u. Laudenschlager, D.: Massenimportieren und -exportieren von XML-Dokumenten, 2023. <https://learn.microsoft.com/de-de/sql/relational-databases/import-export/examples-of-bulk-import-and-export-of-xml-documents-sql-server>. aktualisiert am: 03.03.2023, abgerufen am: 06.03.2023
- [64] West, R., Roth, J., Raghavs, Ray, M., Coulter, D., Sherer, T., Masha, Mauri, D., Loh, S., Rabeler, C., Popovic, J., Guyer, C., Shujia, Fonseca, S. u. Laudenschlager, D.: Beispiele für Massenzugriff auf Daten in Azure Blob Storage, 2023. <https://learn.microsoft.com/de-de/sql/relational-databases/import-export/examples-of-bulk-access-to-data-in-azure-blob-storage>. aktualisiert am: 03.03.2023, abgerufen am: 06.03.2023
- [65] Red Hat: Was ist ein Webhook?, 2022. <https://www.redhat.com/de/topics/automation/what-is-a-webhook>. aktualisiert am: 01.06.2022, abgerufen am: 06.03.2023
- [66] Sahni, D. L.: Difference between Webhook and API Polling, 2022. <https://stories.salesforce-codex.com/2022/09/interview-tips/difference-between-webhook-and-api-polling/>. aktualisiert am: 13.09.2022, abgerufen am: 03.10.2023
- [67] Red Hat: Was ist eine API?, 2023. <https://www.redhat.com/de/topics/api/what-are-application-programming-interfaces>. aktualisiert am: 02.02.2023, abgerufen am: 06.03.2023
- [68] Mohanan, S.: Setting up Oracle CDC (Change Data Capture). 2 Easy Methods, 2023. <https://hevodata.com/learn/oracle-cdc/>. aktualisiert am: 25.09.2023, abgerufen am: 03.10.2023
- [69] Ghanayem, M., West, R., Roth, J., Assaf, W., Julie, Coulter, D., Ray, M., Sherer, T., Engel, D., Cai, S., Guyer, C. u. Rabeler, C.: SQL Server Native Client (ODBC), 2023. <https://learn.microsoft.com/en-us/sql/relational-databases/native-client/odbc/sql-server-native-client-odbc>. aktualisiert am: 03.01.2023, abgerufen am: 06.03.2023
- [70] Geeks for Geeks: PL/SQL Introduction, 2018. <https://www.geeksforgeeks.org/plsql-introduction/>. aktualisiert am: 03.04.2018, abgerufen am: 06.03.2023
- [71] Rapid API: What is an API wrapper? <https://rapidapi.com/blog/api-glossary/api-wrapper/>, abgerufen am: 03.10.2023
- [72] Wiederhold, G.: Mediators in the architecture of future information systems. Computer 25 (1992) 3 SS. 38–49. <https://ieeexplore.ieee.org/document/121508>, abgerufen am: 03.10.2023
- [73] BSI 200-2:2017-10. *Managementsysteme für Informationssicherheit*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_2.pdf?__blob=publicationFile&v=2, abgerufen am: 01.10.2023
- [74] Bundesamt für Sicherheit in der Informationstechnik: Redundanz, Modularität, Skalierbarkeit, 2018. <https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/RZ-Sicherheit/redundanz-modularitaet-skalierbarkeit.pdf>, abgerufen am: 03.10.2023
- [75] BSI 200-3:2017-10. *Risikoanalyse auf der Basis von IT-Grundschutz*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_3.pdf?__blob=publicationFile&v=2, abgerufen am: 01.10.2023

- [76] BSI-Standard 200-4:2023-05. *Business Continuity Management*.
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/BSI_Standards/standard_200_4.pdf?__blob=publicationFile&v=8, abgerufen am: 01.10.2023
- [77] ISO/IEC 2382:2015:2015. *Informaiton technology - Vocabulary*.
<https://www.iso.org/obp/ui/en/#iso:std:iso-iec:2382:ed-1:v2:en>, abgerufen am: 03.10.2023
- [78] AWS: What's The Difference Between Throughput and Latency?, 2023. <https://aws.amazon.com/compare/the-difference-between-throughput-and-latency/>, abgerufen am: 01.10.2023
- [79] Püschmann, N.: Mit Matrizen bewerten. Die perfekte Entscheidungsmatrix, 2021.
<https://blogs.zeiss.com/digital-innovation/de/die-perfekte-entscheidungsmatrix/>. aktualisiert am: 15.11.2021, abgerufen am: 11.02.2023
- [80] Malinverno, P. u. O'Neill, M.: Magic Quadrant for Full Life Cycle API Management, 2018.
<https://www.gartner.com/reviews/market/full-life-cycle-api-management>, abgerufen am: 23.09.2018
- [81] Oracle Deutschland: PL/SQL für Entwickler. <https://www.oracle.com/de/database/technologies/app-dev/plsql.html#:~:text=PL%2FSQL%20ist%20eine%20prozedurale,und%20in%20der%20Datenbank%20gespeichert.>, abgerufen am: 19.08.2023
- [82] Bringmann, B. Dr, Beierschoder, M., Fach, P., Esser, R., Wendland, P. u. Göckelmann, S.: Fueling the AI transformation - Four key actions powering widespread value from AI, right now in Germany. Deloitte's State of AI in the Enterprise. Deloitte's State of AI in the Enterprise, 2022. [Web] <https://www2.deloitte.com/content/dam/Deloitte/de/Documents/Innovation/Deloitte-State-of-AI-2022.pdf>. aktualisiert am: December 2022, abgerufen am: 12.09.2023
- [83] Charboneau, T.: Breaking Down SmartBear's 2020 State of API Report, 2020. <https://nordicapis.com/breaking-down-smartbears-2020-state-of-api-report/>. aktualisiert am: 27.08.2020, abgerufen am: 12.09.2023
- [84] Schaffry, A.: No-code/Low-code Studie 2022, München 2022.
https://25084391.fs1.hubspotusercontent-eu1.net/hubfs/25084391/Sustainability_Content/Whitepaper%20u.%20Leitfaden/No-Code_Low-Code-Studie-CIO_Computerwoche_2022.pdf. aktualisiert am: Januar 2023, abgerufen am: 12.09.2023
- [85] European Central Bank: The revised Payment Services Directive (PSD2) and the transition to stronger payments security, 2018. https://www.ecb.europa.eu/paym/intro/mip-online/2018/html/1803_revisedpsd.en.html. aktualisiert am: 12.09.2023, abgerufen am: 03.10.2023
- [86] CDISC: ADaM - Analysis Data Model, 2021. <https://www.cdisc.org/standards/foundational/adam>. aktualisiert am: 29.11.2021, abgerufen am: 12.09.2023
- [87] CDISC: SDTM - Study Data Tabulation Model, 2021. <https://www.cdisc.org/standards/foundational/sdtm>. aktualisiert am: 29.11.2021, abgerufen am: 12.09.2023
- [88] CDISC: CDISC Standards in the Clinical Research Process. <https://www.cdisc.org/standards>, abgerufen am: 03.10.2023
- [89] 2018. 16739. *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries*. <https://www.iso.org/standard/70303.html>, abgerufen am: 12.09.2013

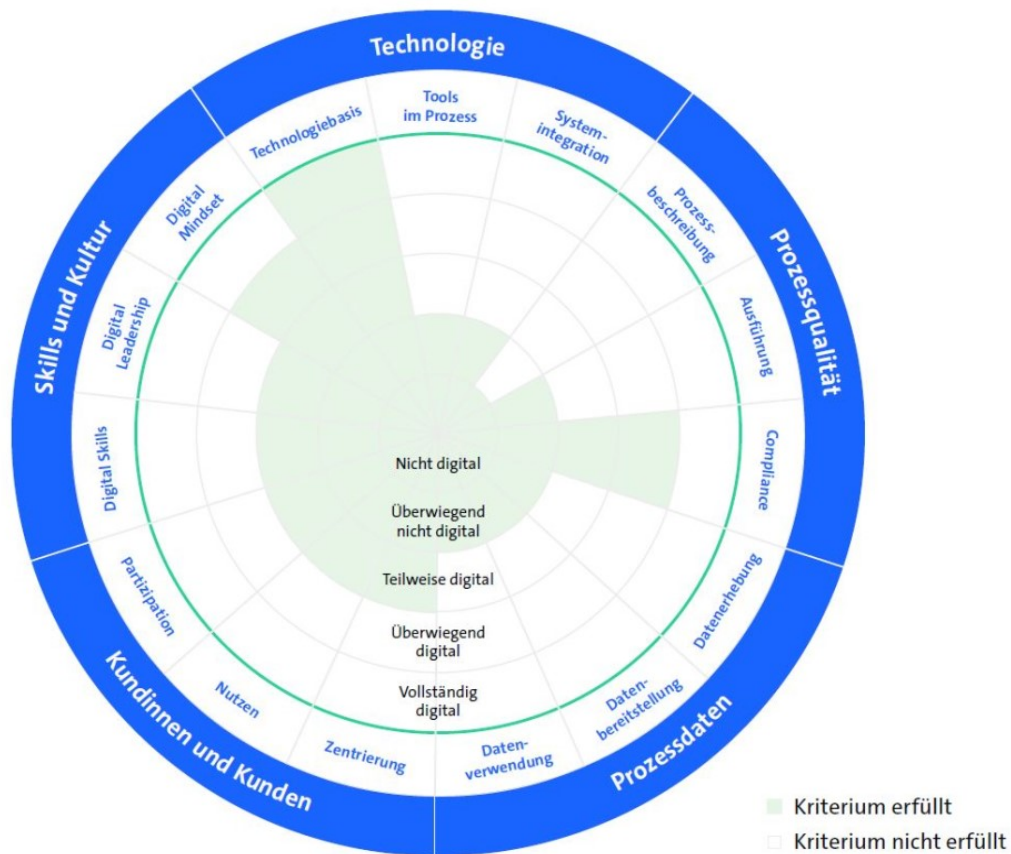
Anhang

Anhang 1 Auszug aus der IW Studie „Digitalisierungsindex 2022“



Ergebnisse des Digitalisierungsindex nach Branchen in Indexpunkten; Erhebungsjahre 2020, 2021 und 2022.
(Quelle: Institut der deutschen Wirtschaft [4])

Anhang 2 Bitkom Reifegradmodell Digitale Prozesse 2.0



Beispielhafte Darstellung des Reifegradmodells Digitale Prozesse 2.0 der Bitkom [15]

Dimension	Kriterium	Operationalisierung
Technologie	Technologiebasis	1. Alle eingehenden Informationen für den Prozess sind vollständig digital. 2. Alle ausgehenden Informationen für den Prozess sind vollständig digital.
	Tools im Prozess	3. Es wird eine Software-Lösungen zur vollständigen Modellierung und Analyse des Geschäftsprozesses eingesetzt. 4. Der Geschäftsprozess ist mit einer Software-Lösung vollständig automatisiert.
	Systemintegration	5. Alle im Prozess verwendeten Software-Lösungen sind vollständig integriert. 6. Der Prozess läuft vollständig ohne unnötige Medienbrüche.

Auszug aus dem Kriterienkatalog zur Dimension „Technologie“ des Reifegradmodells Digitale Prozesse 2.0
(Quelle: Bitkom [15])

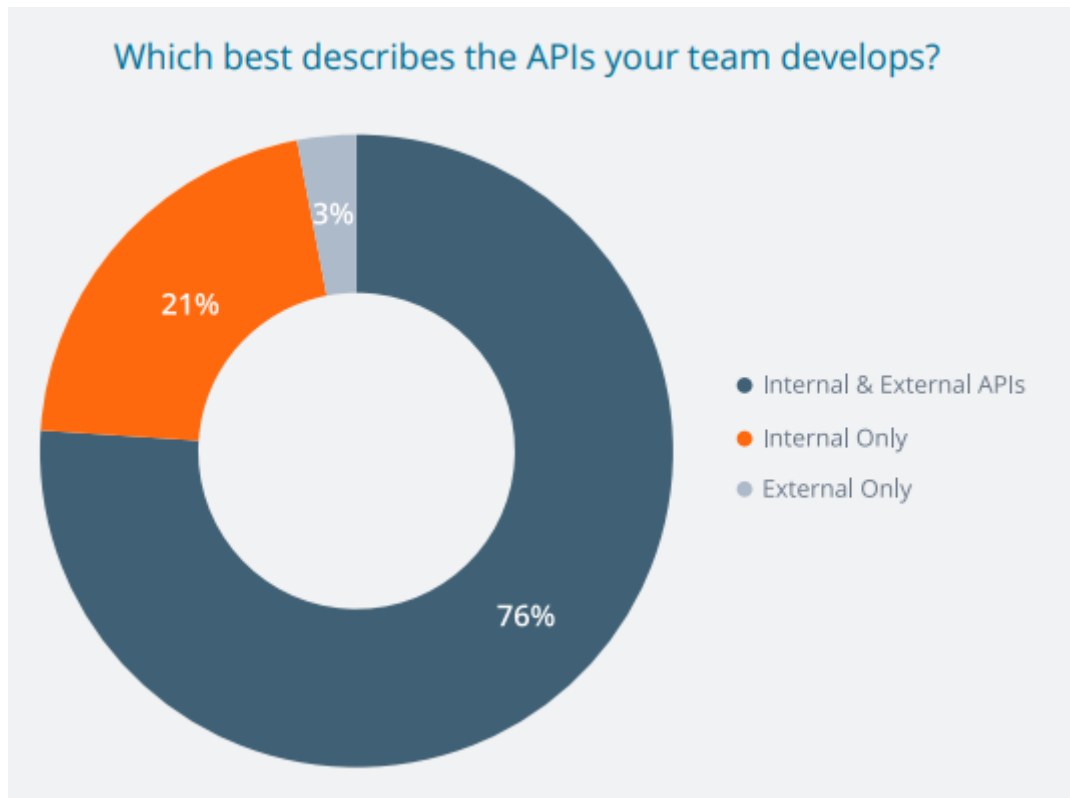
Anhang 3 Auszug aus der SmartBear Erhebung „The State of API“



*Beweggründe für API-fizierung
(Quelle: SmartBear [38])*



*Etablierung von APIs im Unternehmen
(Quelle: SmartBear [38])*

Etablierung von APIs im Unternehmensumfeld [38]

*Akteure im API-Management
(Quelle: SmartBear [38])*

8. Standardization is the #1 API technology challenge facing API teams

As organizations scale their API programs, the need to establish standards for how APIs are developed and maintained within that organization becomes clear. This is especially true with the emergence of new software architecture formats, like microservices, where organizations can be maintaining hundreds or even thousands of different APIs within their service architecture.

*Standardisierung als Herausforderung für API-fizierung
(Quelle: SmartBear [38])*

Anhang 5 Thementag API-Security - Berliner Forum Digitalisierung

Berliner Forum Digitalisierung
Hochschule und Praxis im Dialog

Thementag API-Security

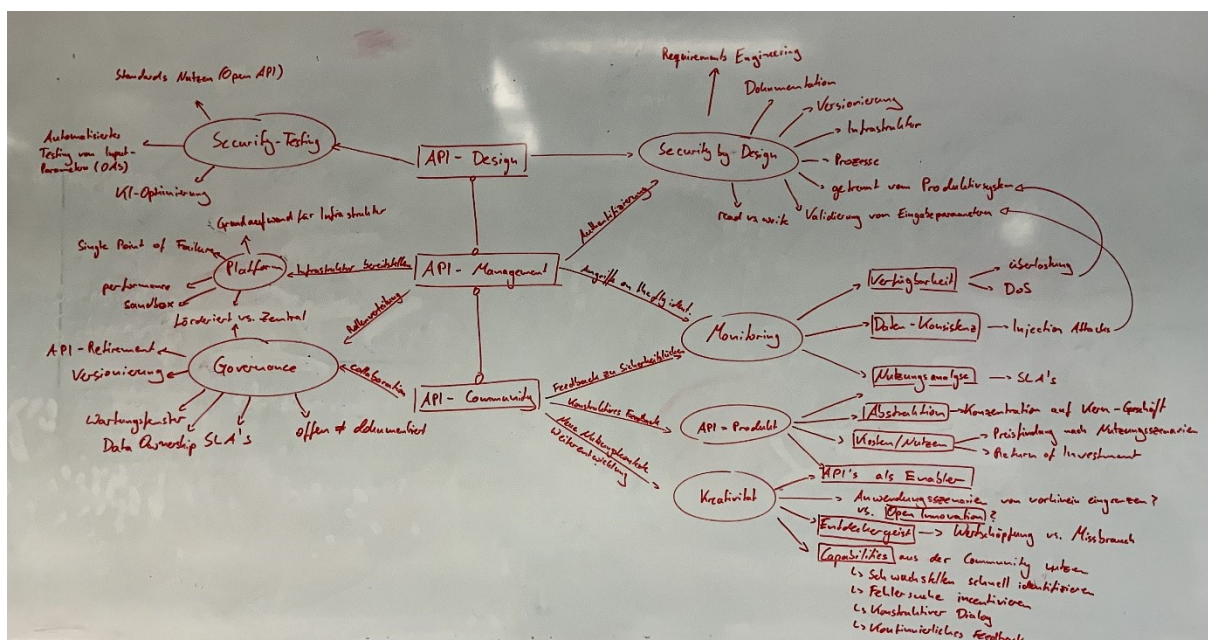
19. September 2019

13.30–18.00 Uhr

Campus Lichtenberg, Haus 6A, Raum 014

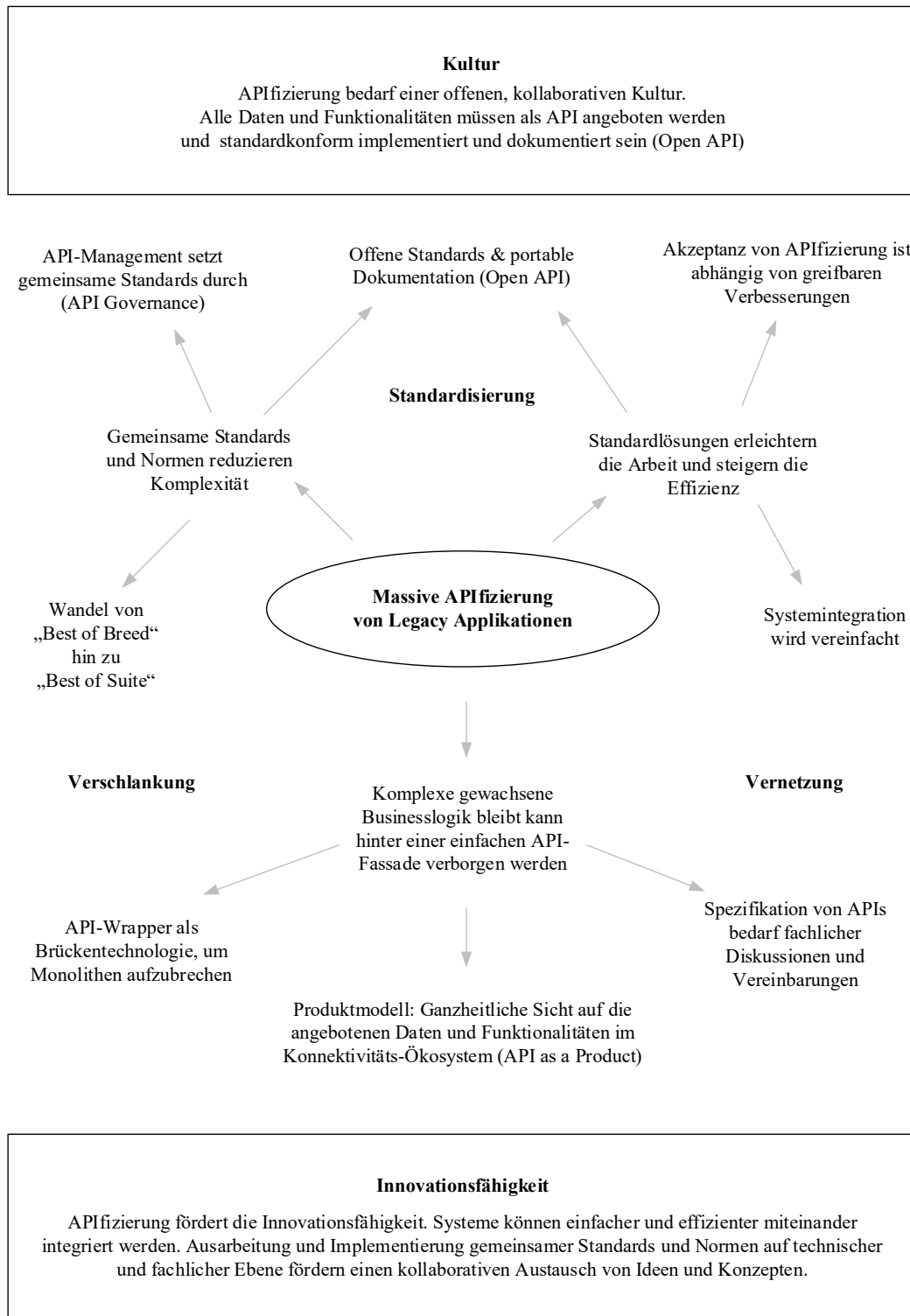
- 13.00–13.30 Uhr **Willkommen und Eröffnung**
Prof. Dr. Andreas Schmietendorf (HWR Berlin)
- 13.30–14.15 Uhr **Keynote: »Herausforderungen der Digitalisierung mit openData/ openAPI begegnen«**
Michael Binzen (Bitkom/DB)
- 14.30–16.15 Uhr **World Cafes**
Angriffsszenarien und Risiken bei open Data/open API
Fraud detection, KI-gestützte Angriffe
Sicherheit durch betriebliche Maßnahmen
Data Ownership, Governance, Digitalisierungsstrategie
Sicherheit durch Technische Maßnahmen
Architektur, Analytik, Messen
API-Management
Konzepte, Plattformen, API's im Praxiseinsatz
- 16.15–17.00 Uhr **Keynote: »Secure open Data: Identity Access Management mit Blockchain«**
Gabriel Landa (Alpha-Gen)
- 17.00–17.45 Uhr **Keynote: »Ausblick und Faszination: AI über APIs einbinden«**
Anja Fiegler (Microsoft)

Agenda des Thementags „API-Security“ des Berliner Forum Digitalisierung am 19. September 2019



Ergebnisse einer Expertendiskussion zu API-Management im Rahmen des Thementags „API-Security“
des Berliner Forum Digitalisierung am 19. September 2019
(Quelle: Eigenes Foto)

Anhang 6 Expertendiskussion zu API-fizierung



*Ergebnisse der Expertendiskussion im Workshop "Evaluation of Service-APIs – ESAPI 2020"
(Quelle: Eigener Beitrag im veröffentlichten Tagungsbericht [22])*

Anhang 7 Chat-GPT Anfrage zu Architekturmustern für Systemintegration



Beschreibe und klassifiziere die gängigen Architekturmuster für Systemintegration.



Ein Architekturmuster für Systemintegration beschreibt eine bewährte Methode, um verschiedene Systeme miteinander zu integrieren und zu verbinden. Hier sind einige der gängigsten Architekturmuster für Systemintegration:



1. Enterprise Service Bus (ESB): Dies ist eine Middleware-Plattform, die als zentraler Integrationspunkt für verschiedene Systeme dient. Der ESB übernimmt die Verwaltung der Kommunikation und Übertragung von Daten zwischen den Systemen und stellt eine einheitliche Schnittstelle für die Kommunikation bereit.
2. Microservices-Architektur: Dies ist ein Ansatz zur Integration von Systemen, bei dem jedes System als unabhängiger Microservice entwickelt wird, der eine spezifische Funktionalität bereitstellt. Die Microservices kommunizieren miteinander über eine offene API.
3. Event-Driven Architecture (EDA): Dies ist ein Ansatz, bei dem Systeme über Ereignisse miteinander kommunizieren und Daten austauschen. Ein Ereignis kann z.B. ein neues Datum in einer Datenbank sein oder eine Änderung an einem bestimmten Datensatz.
4. Point-to-Point-Integration: Dies ist ein Ansatz, bei dem Systeme direkt miteinander verbunden werden, um Daten auszutauschen. Diese Art der Integration ist am einfachsten zu implementieren, aber auch am schwierigsten zu verwalten, da Änderungen an einem System oft Auswirkungen auf alle verbundenen Systeme haben.
5. API-basierte Integrationsarchitektur: Dies ist ein Ansatz, bei dem Systeme mithilfe von APIs miteinander integriert werden. Eine API stellt eine programmierbare Schnittstelle bereit, über die Systeme Daten austauschen können.

Jedes dieser Architekturmuster hat seine eigenen Stärken und Schwächen und die Wahl des richtigen Musters hängt von den spezifischen Anforderungen und Bedürfnissen des Integrationsprojekts ab.

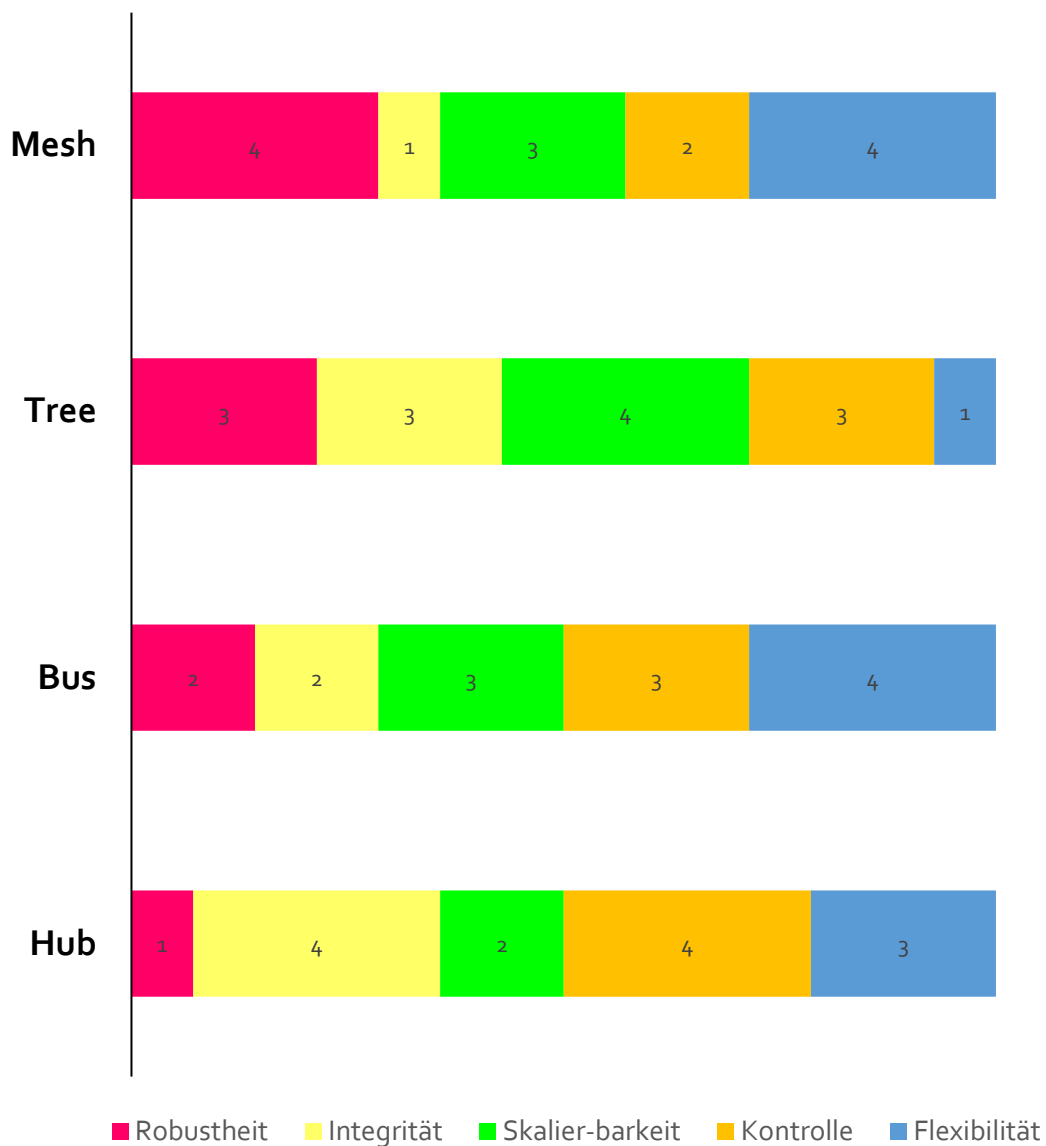
*Antwort von Chat GPT auf gängige Architekturmuster
(Quelle: Eigener Screenshot von open.ai ChatGPT 3.5)*

Anhang 8 Ergebnisse der Gegenüberstellung von Integrationstopologien

	Hub	Bus	Tree	Mesh
Robustheit	sehr schwach (1)	schwach (2)	stark (3)	sehr stark (4)
Integrität	sehr stark (4)	schwach (2)	stark (3)	sehr schwach (1)
Skalierbarkeit	schwach (2)	stark (3)	sehr stark (4)	stark (3)
Kontrolle	sehr stark (4)	stark (3)	stark (3)	schwach (2)
Flexibilität	stark (3)	sehr stark (4)	sehr schwach (1)	sehr stark (4)
Summe	14	14	14	14

Skala: sehr schwach (1), über schwach (2), stark (3), bis sehr stark (4)

Tabellarische Darstellung der Ergebnisse der Gegenüberstellung von Integrationstopologien



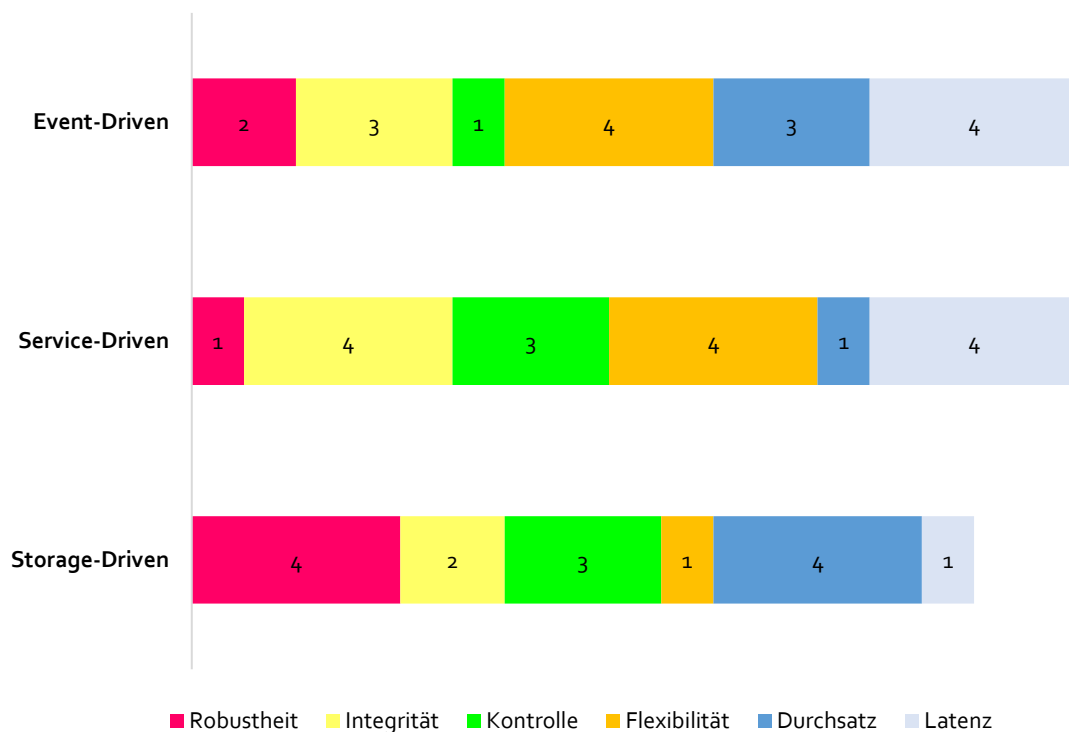
Grafische Darstellung der Ergebnisse der Gegenüberstellung von Integrationstopologien

Anhang 9 Ergebnisse der Gegenüberstellung von Integrationsparadigmen

	Storage-Driven	Service-Driven	Event-Driven
Robustheit	sehr stark (4)	sehr schwach (1)	schwach (2)
Integrität	schwach (2)	sehr stark (4)	stark (3)
Kontrolle	stark (3)	stark (3)	sehr schwach (1)
Flexibilität	sehr schwach (1)	sehr stark (4)	sehr stark (4)
Durchsatz	sehr stark (4)	sehr schwach (1)	stark (3)
Latenz	sehr schwach (1)	sehr stark (4)	sehr stark (4)
Summe	15	16	17

Skala: sehr schwach (1), über schwach (2), stark (3), bis sehr stark (4)

Tabellarische Darstellung der Ergebnisse der Gegenüberstellung von Integrationsparadigmen



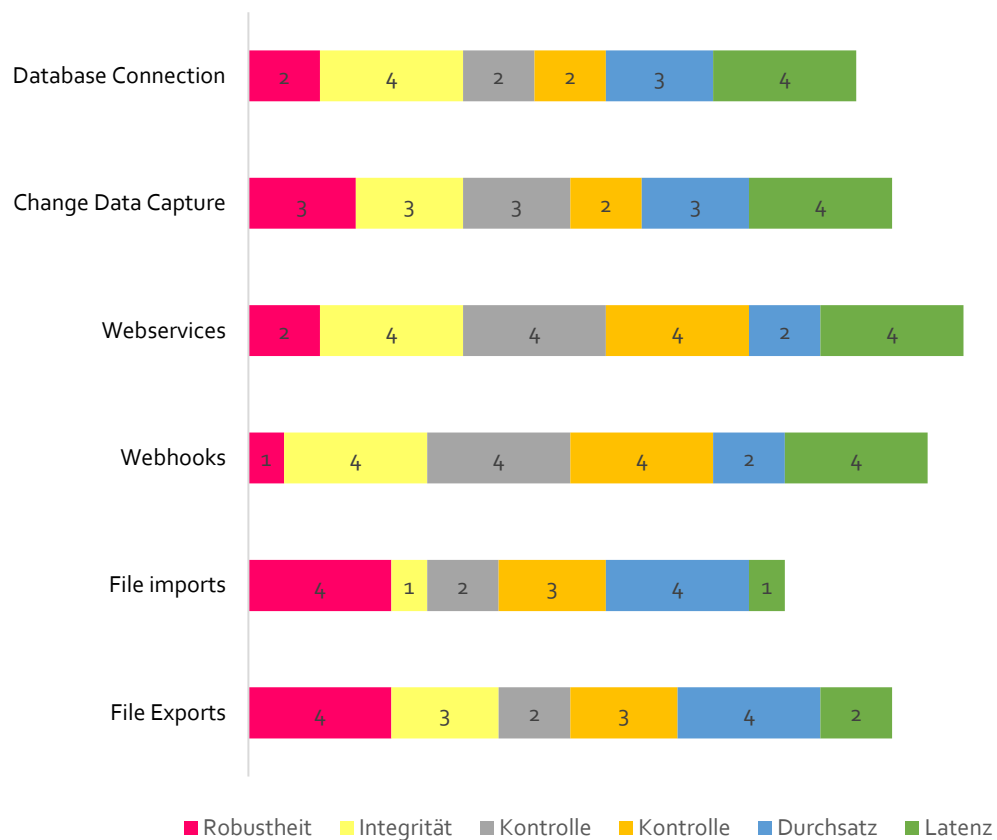
Grafische Darstellung der Ergebnisse der Gegenüberstellung von Integrationsparadigmen

Anhang 10 Ergebnisse der Gegenüberstellung von Integrationsmethoden

	File Exports	File Imports	Web-hooks	Web-services	Change Data Capture	Database-Connection
Robustheit	sehr stark (4)	sehr stark (4)	sehr schwach (1)	schwach (2)	sehr stark (4)	schwach (2)
Integrität	stark (3)	sehr schwach (1)	sehr stark (4)	sehr stark (4)	stark (3)	stark (3)
Kontrolle	schwach (2)	schwach (2)	sehr stark (4)	sehr stark (4)	stark (3)	schwach (2)
Flexibilität	stark (3)	stark (3)	sehr stark (4)	sehr stark (4)	schwach (2)	schwach (2)
Durchsatz	sehr stark (4)	sehr stark (4)	schwach (2)	schwach (2)	stark (3)	stark (3)
Latenz	schwach (2)	sehr schwach (1)	sehr stark (4)	sehr stark (4)	sehr stark (4)	sehr stark (4)
Summe	18	15	19	20	18	16

Skala: sehr schwach (1), über schwach (2), stark (3), bis sehr stark (4)

Tabellarische Darstellung der Ergebnisse der Gegenüberstellung von Integrationsmethoden



Grafische Darstellung der Ergebnisse der Gegenüberstellung von Integrationsmethoden

Anhang 11 Anwendung des GAP-Modells für ein operatives Data Warehouse

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Gewichtung	Ziel-Bewertung
Robustheit	-	2	3	2	2	2	3	14 17%	3
Integrität	2	-	3	2	3	1	3	14 17%	3
Skalierbarkeit	1	1	-	2	3	1	2	10 12%	2
Kontrolle	2	2	2	-	2	1	1	10 12%	2
Flexibilität	2	2	1	2	-	1	2	10 12%	2
Durchsatz	2	2	3	3	3	-	2	15 18%	3
Latenz	1	1	2	3	2	2	-	11 13%	2
Skala: 1 = unwichtiger 2 = gleich wichtig 3 = wichtiger 84 100%									

Gewichtung der Kriterien für die Integration eines operativen Data Warehouse

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Gewichtung	17%	17%	12%	12%	12%	18%	13%	
Ziel	3	3	2	2	2	3	2	
Hub	1	4	2	4	3	-	-	1,90
Bus	2	2	3	3	4	-	-	1,86
Tree	3	3	4	3	1	-	-	1,95
Mesh	4	1	3	2	4	-	-	1,90
Storage-Driven	4	2	-	3	1	4	1	2,13
Service-Driven	1	4	-	3	4	1	4	2,32
Event-Driven	2	3	-	1	4	3	4	2,35
File-Exports (Mediator)	1	3	-	2	3	4	2	2,05
File-Imports	4	1	-	2	3	4	1	2,08
Webhooks (Wrapper)	1	4	-	4	4	2	4	2,57
Webservices	2	4	-	4	4	2	4	2,74
Change Data Capture (Wrapper)	1	3	-	3	2	3	4	2,18
Database Connection	2	4	-	2	2	3	4	2,39

Skala: 1 = sehr schwach 2 = schwach 3 = stark 4 = sehr stark

Optionsraum für das Integrationsdesign eines operativen Data Warehouse

Topologie	Paradigma	Methode	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Kriterien Erfüllt	Bewertung
Bus	Storage-Driven	Webservices	3	3	3	3	3	3	3	7	6,73
Bus	Storage-Driven	Database-Connection	3	3	3	3	2	4	3	7	6,38
Tree	Storage-Driven	File-Exports	3	3	4	3	2	4	2	7	6,13
Tree	Storage-Driven	Webhooks	3	3	4	3	2	3	3	7	6,65
Tree	Storage-Driven	Webservices	3	3	4	3	2	3	3	7	6,82
Tree	Service-Driven	File-Imports	3	3	4	3	3	3	3	7	6,36
Hub	Storage-Driven	File-Exports	2	3	2	3	2	4	2	6	6,08
Hub	Storage-Driven	Webhooks	2	3	2	4	3	3	3	6	6,61
Hub	Storage-Driven	Webservices	2	3	2	4	3	3	3	6	6,77
Hub	Storage-Driven	Change-Data-Capture	2	3	2	3	2	4	3	6	6,21
Hub	Storage-Driven	Database-Connection	2	3	2	3	2	4	3	6	6,43
Hub	Service-Driven	File-Exports	1	4	2	3	3	3	3	6	6,27
Hub	Service-Driven	File-Imports	2	3	2	3	3	3	3	6	6,31
Bus	Storage-Driven	Webhooks	2	3	3	3	3	3	3	6	6,56
Bus	Service-Driven	File-Exports	1	3	3	3	4	3	3	6	6,23
Tree	Storage-Driven	Change-Data-Capture	3	3	4	3	1	4	3	6	6,26
Tree	Storage-Driven	Database-Connection	3	3	4	3	1	4	3	6	6,48
Tree	Service-Driven	File-Exports	2	3	4	3	3	3	3	6	6,32
Mesh	Storage-Driven	File-Exports	3	2	3	2	3	4	2	6	6,08
Mesh	Storage-Driven	Webhooks	3	2	3	3	3	3	3	6	6,61
Mesh	Storage-Driven	Webservices	3	2	3	3	3	3	3	6	6,77
Mesh	Storage-Driven	Change-Data-Capture	3	2	3	3	2	4	3	6	6,21
Mesh	Storage-Driven	Database-Connection	3	2	3	2	2	4	3	6	6,65
Mesh	Service-Driven	File-Exports	2	3	3	2	4	3	3	6	6,27
Mesh	Service-Driven	File-Imports	3	2	3	2	4	3	3	6	6,31
Hub	Storage-Driven	File-Imports	3	2	2	3	2	4	1	5	6,12
Hub	Service-Driven	Webhooks	1	4	2	4	4	2	4	5	6,80
Hub	Service-Driven	Webservices	1	4	2	4	4	2	4	5	6,96
Hub	Service-Driven	Change-Data-Capture	1	4	2	3	3	2	4	5	6,40
Hub	Service-Driven	Database-Connection	1	4	2	3	3	2	4	5	6,62
Bus	Storage-Driven	File-Exports	2	2	3	3	3	4	2	5	6,04
Bus	Storage-Driven	File-Imports	3	2	3	3	3	4	1	5	6,07
Bus	Storage-Driven	Change-Data-Capture	2	2	3	3	2	4	3	5	6,17
Bus	Service-Driven	File-Imports	2	2	3	3	4	3	3	5	6,26
Bus	Service-Driven	Webhooks	1	3	3	3	4	2	4	5	6,75
Bus	Service-Driven	Webservices	2	3	3	3	4	2	4	5	6,92
Bus	Service-Driven	Change-Data-Capture	1	3	3	3	3	2	4	5	6,36

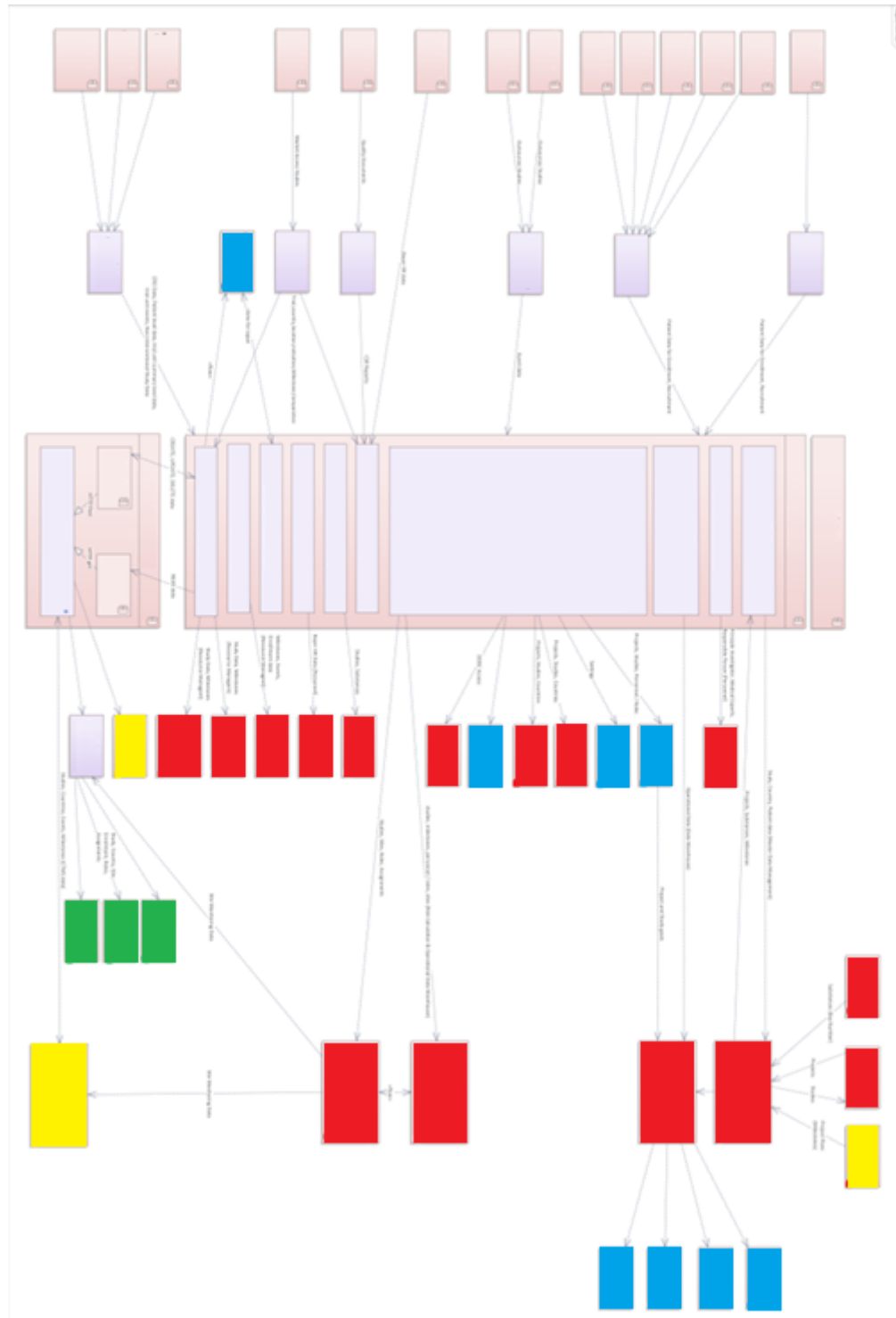
Topologie	Paradigma	Methode	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Kriterien Erfüllt	Bewertung
Bus	Service-Driven	Database-Connection	2	3	3	3	3	2	4	5	6,57
Tree	Storage-Driven	File-Imports	4	2	4	3	2	4	1	5	6,17
Tree	Service-Driven	Webhooks	2	4	4	3	3	2	4	5	6,85
Tree	Service-Driven	Webservices	2	4	4	3	3	2	4	5	7,01
Tree	Service-Driven	Change-Data-Capture	2	3	4	3	2	2	4	5	6,45
Tree	Service-Driven	Database-Connection	2	4	4	3	2	2	4	5	6,67
Mesh	Storage-Driven	File-Imports	4	1	3	2	3	4	1	5	6,12
Mesh	Service-Driven	Webhooks	2	3	3	3	4	2	4	5	6,80
Mesh	Service-Driven	Webservices	2	3	3	3	4	2	4	5	6,96
Mesh	Service-Driven	Change-Data-Capture	2	3	3	3	3	2	4	5	6,40
Mesh	Service-Driven	Database-Connection	2	3	3	2	3	2	4	5	6,62

Ergebnisse des GAP-Modells für die Integration eines operativen Data Warehouse

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Anforderungsprofil	4	3	1	2	1	3	1	
Option A1: Storage-Driven Webservices in Tree	3	3	4	3	2	3	3	6,82
Option B: Storage-Driven Webservices in Bus	3	3	3	3	3	3	3	6,73
Option A: Option: Storage-Driven Webhooks in Tree	3	3	4	3	2	3	3	6,65
Option D: Storage-Driven Database Connection in Bus	3	3	3	3	2	4	3	6,38
Option C: Service-Driven File-Imports in Tree	3	3	4	3	3	3	3	6,36
Option E: Storage-Driven File-Exports in Tree	3	3	4	3	2	4	2	6,13
Implementierung: Storage-Driven File-Import in Integration Tree	4	2	4	3	2	4	1	6,82
Implementierung: Storage-Driven Webservices in Integration Tree	3	3	4	3	2	3	3	6,48
Implementierung: Storage-Driven Database-Connection in Integration Tree	3	3	4	3	1	4	3	6,17

Ergebnisse des GAP-Modells für Fallstudie 1

Anhang 12 Modell einer historisch gewachsenen Systemlandschaft in Fallstudie 2



Rot: Satellit 1 - Archetyp: Storage-Driven Database-Connection in Integration Tree (15 Systeme)

Blau: Satellit 2 - Archetyp: Service-Driven Database-Connection in Integration Tree (8 Systeme)

Gelb: Satellit 3 - Archetyp: Service-Driven Webservice in Integration Tree (2 Systeme)

Grün: Satellit 4 - Archetyp: Service-Driven Webservice in Integration Tree (3 Systeme)

*Modell der Integrationsarchitektur des betrachteten Anwendungsfalls in Fallstudie 2 (operatives Kernsystem)
(Quelle: Eigene Darstellung, Inhalte anonymisiert)*

Anhang 13 Anwendung des GAP-Modells für die Anbindung von Satelliten

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Gewichtung		Ziel-Bewertung
Robustheit	-	1	3	1	1	2	1	9	10%	1
Integrität	3	-	3	2	2	3	2	15	17%	3
Skalierbarkeit	1	1	-	1	1	2	1	7	8%	1
Kontrolle	3	2	3	-	3	3	3	17	20%	4
Flexibilität	3	3	3	1	-	2	1	13	15%	2
Durchsatz	2	2	2	1	2	-	1	10	11%	2
Latenz	3	3	3	1	3	3	-	16	18%	3
Skala: 1 = unwichtiger 2 = gleich wichtig 3 = wichtiger									87	100%

Gewichtung der Kriterien für die Anbindung von Satellitensystemen an ein operatives Kernsystem

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Gewichtung	10%	17%	8%	20%	15%	11%	18%	
Ziel	1	3	1	4	2	2	3	
Hub	1	4	2	4	3	-	-	2,18
Bus	2	2	3	3	4	-	-	1,98
Tree	3	3	4	3	1	-	-	1,89
Mesh	4	1	3	2	4	-	-	1,82
Storage-Driven	4	2	-	3	1	4	1	2,41
Service-Driven	1	4	-	3	4	1	4	2,90
Event-Driven	2	3	-	1	4	3	4	2,80
File-Exports (Mediator)	1	3	-	2	3	4	2	2,56
File-Imports	4	1	-	2	3	4	1	2,34
Webhooks	1	4	-	4	4	2	4	3,28
Webservices (Wrapper)	1	4	-	4	4	2	4	3,28
Change-Data-Capture (Wrapper)	1	3	-	3	2	3	4	2,79
Database-Connection (Wrapper)	2	4	-	2	2	3	4	2,87

Skala: 1 = sehr schwach 2 = schwach 3 = stark 4 = sehr stark

Optionsraum für das Integrationsdesign der Anbindung von Satellitensystemen an ein operatives Kernsystem

Topologie	Paradigma	Methode	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Kriterien Erfüllt	Bewertung
Hub	Storage-Driven	Webhooks	2	3	2	4	3	3	3	7	7,87
Hub	Storage-Driven	Webservices	2	3	2	4	3	3	3	7	7,87
Hub	Service-Driven	Webhooks	1	4	2	4	4	2	4	7	8,36
Hub	Service-Driven	Webservices	1	4	2	4	4	2	4	7	8,36
Hub	Event-Driven	Webhooks	1	4	2	4	4	2	4	7	8,36
Hub	Event-Driven	Webservices	1	4	2	4	4	2	4	7	8,36
Hub	Storage-Driven	Change-Data-Capture	2	3	2	3	2	4	3	6	7,39
Hub	Storage-Driven	Database-Connection	2	3	2	3	2	4	3	6	7,47
Hub	Service-Driven	File-Exports	1	4	2	3	3	3	3	6	7,64
Hub	Service-Driven	File-Imports	2	3	2	3	3	3	3	6	7,43
Hub	Service-Driven	Change-Data-Capture	1	4	2	3	3	2	4	6	7,87
Hub	Service-Driven	Database-Connection	1	4	2	3	3	2	4	6	7,95
Hub	Event-Driven	File-Exports	1	4	2	3	3	3	3	6	7,64
Hub	Event-Driven	File-Imports	2	3	2	3	3	3	3	6	7,43
Hub	Event-Driven	Change-Data-Capture	1	4	2	3	3	2	4	6	7,87
Hub	Event-Driven	Database-Connection	1	4	2	3	3	2	4	6	7,95
Bus	Storage-Driven	Webhooks	2	3	3	3	3	3	3	6	7,67
Bus	Storage-Driven	Webservices	2	3	3	3	3	3	3	6	7,67
Bus	Storage-Driven	Database-Connection	3	3	3	3	2	4	3	6	7,26
Bus	Service-Driven	File-Exports	1	3	3	3	4	3	3	6	7,44
Bus	Service-Driven	Webhooks	1	3	3	3	4	2	4	6	8,15
Bus	Service-Driven	Webservices	1	3	3	3	4	2	4	6	8,15
Bus	Service-Driven	Change-Data-Capture	1	3	3	3	3	2	4	6	7,67
Bus	Service-Driven	Database-Connection	2	3	3	3	3	2	4	6	7,75
Bus	Event-Driven	File-Exports	2	3	3	2	4	4	3	6	7,44
Bus	Event-Driven	Webhooks	2	3	3	3	4	3	4	6	8,15
Bus	Event-Driven	Webservices	2	3	3	3	4	3	4	6	8,15
Bus	Event-Driven	Change-Data-Capture	2	3	3	2	3	3	4	6	7,67
Bus	Event-Driven	Database-Connection	2	3	3	2	3	3	4	6	7,75
Tree	Storage-Driven	Webhooks	3	3	4	3	2	3	3	6	7,57
Tree	Storage-Driven	Webservices	3	3	4	3	2	3	3	6	7,57
Tree	Service-Driven	File-Exports	2	3	4	3	3	3	3	6	7,34
Tree	Service-Driven	File-Imports	3	3	4	3	3	3	3	6	7,13
Tree	Service-Driven	Webhooks	2	4	4	3	3	2	4	6	8,06
Tree	Service-Driven	Webservices	2	4	4	3	3	2	4	6	8,06
Tree	Service-Driven	Change-Data-Capture	2	3	4	3	2	2	4	6	7,57
Tree	Service-Driven	Database-Connection	2	4	4	3	2	2	4	6	7,66
Tree	Event-Driven	File-Exports	2	3	4	2	3	4	3	6	7,34
Tree	Event-Driven	Webhooks	2	3	4	3	3	3	4	6	8,06

Topologie	Paradigma	Methode	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Kriterien Erfüllt	Bewertung
Tree	Event-Driven	Webservices	2	3	4	3	3	3	4	6	8,06
Tree	Event-Driven	Change-Data-Capture	2	3	4	2	2	3	4	6	7,57
Tree	Event-Driven	Database-Connection	2	3	4	2	2	3	4	6	7,66
Mesh	Service-Driven	File-Exports	2	3	3	2	4	3	3	6	7,28
Mesh	Service-Driven	Webhooks	2	3	3	3	4	2	4	6	7,99
Mesh	Service-Driven	Webservices	2	3	3	3	4	2	4	6	7,99
Mesh	Service-Driven	Change-Data-Capture	2	3	3	3	3	2	4	6	7,51
Mesh	Service-Driven	Database-Connection	2	3	3	2	3	2	4	6	7,59
Mesh	Event-Driven	Webhooks	2	3	3	2	4	3	4	6	7,99
Mesh	Event-Driven	Webservices	2	3	3	2	4	3	4	6	7,99
Mesh	Event-Driven	Database-Connection	3	3	3	2	3	3	4	6	7,59
Hub	Storage-Driven	File-Exports	2	3	2	3	2	4	2	5	7,16
Bus	Storage-Driven	Change-Data-Capture	2	2	3	3	2	4	3	5	7,18
Bus	Service-Driven	File-Imports	2	2	3	3	4	3	3	5	7,22
Bus	Event-Driven	File-Imports	3	2	3	2	4	4	3	5	7,22
Tree	Storage-Driven	File-Exports	3	3	4	3	2	4	2	5	6,86
Tree	Storage-Driven	Change-Data-Capture	3	3	4	3	1	4	3	5	7,09
Tree	Storage-Driven	Database-Connection	3	3	4	3	1	4	3	5	7,17
Tree	Event-Driven	File-Imports	3	2	4	2	3	4	3	5	7,13
Mesh	Storage-Driven	Webhooks	3	2	3	3	3	3	3	5	7,51
Mesh	Storage-Driven	Webservices	3	2	3	3	3	3	3	5	7,51
Mesh	Storage-Driven	Change-Data-Capture	3	2	3	3	2	4	3	5	7,02
Mesh	Storage-Driven	Database-Connection	3	2	3	2	2	4	3	5	7,70
Mesh	Service-Driven	File-Imports	3	2	3	2	4	3	3	5	7,06
Mesh	Event-Driven	File-Exports	2	2	3	2	4	4	3	5	7,28
Mesh	Event-Driven	File-Imports	3	2	3	2	4	4	3	5	7,06
Mesh	Event-Driven	Change-Data-Capture	2	2	3	2	3	3	4	5	7,51
Hub	Storage-Driven	File-Imports	3	2	2	3	2	4	1	4	6,94
Bus	Storage-Driven	File-Exports	2	2	3	3	3	4	2	4	6,95
Bus	Storage-Driven	File-Imports	3	2	3	3	3	4	1	4	6,74
Tree	Storage-Driven	File-Imports	4	2	4	3	2	4	1	4	6,64
Mesh	Storage-Driven	File-Exports	3	2	3	2	3	4	2	4	6,79
Mesh	Storage-Driven	File-Imports	4	1	3	2	3	4	1	4	6,57

Ergebnisse des GAP-Modells für die Anbindung von Satellitensystemen an ein operatives Kernsystem

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Anforderungsprofil	1	3	1	4	2	2	3	
Service-Driven und Event-Driven Webservices in Hub	1	4	2	4	4	2	4	8,36
Storage-Driven Webservices in Hub	2	3	2	4	3	3	3	8,36
Empfehlung: Service-Driven Webservices in Hub	1	4	2	4	4	2	4	8,15
Satellit 1: Storage-Driven Database Connection in Integration Tree	3	3	4	3	1	4	3	6,94
Satellit 2: Service-Driven Database-Connection in Integration Tree	2	4	4	3	2	2	4	7,66
Satellit 3: Service-Driven Webservices in Integration Tree	2	4	4	3	3	2	4	7,47
Satellit 4: Service-Driven Webservices in Integration Hub	1	4	2	4	4	2	4	6,95

Ergebnisse des GAP-Modells für Fallstudie 2

Anhang 14 Anwendung des GAP-Modells für Synchronisierung von Kernsystemen

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz		Gewichtung	Ziel-Bewertung
Robustheit	-	2	3	2	3	3	2	15	18%	3
Integrität	2	-	3	2	3	3	2	15	18%	3
Skalierbarkeit	1	1	-	1	2	1	1	7	8%	1
Kontrolle	2	2	3	-	2	3	2	14	17%	3
Flexibilität	1	1	2	2	-	2	1	9	11%	1
Durchsatz	1	1	3	1	2	-	1	9	11%	1
Latenz	2	2	3	2	3	3	-	15	18%	3
Skala: 1 = unwichtiger 2 = gleich wichtig 3 = wichtiger								84	100%	15

Gewichtung der Kriterien für die Synchronisierung von Kernsystemen

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Gewichtung	18%	18%	8%	17%	11%	11%	18%	
Ziel	3	3	1	3	1	1	3	
Hub	1	4	2	4	3	-	-	2,05
Bus	2	2	3	3	4	-	-	1,89
Tree	3	3	4	3	1	-	-	2,01
Mesh	4	1	3	2	4	-	-	1,90
Storage-Driven	4	2	-	3	1	4	1	2,57
Service-Driven	1	4	-	3	4	1	4	2,71
Event-Driven	2	3	-	1	4	3	4	2,74
File-Exports (Mediator)	1	3	-	2	3	4	2	2,44
File-Imports	4	1	-	2	3	4	1	2,44
Webhooks (Wrapper)	1	4	-	4	4	2	4	3,06
Webservices (Wrapper)	1	4	-	4	4	2	4	3,06
Change-Data-Capture (Wrapper)	1	3	-	3	2	3	4	2,68
Database-Connection	2	4	-	2	2	3	4	2,87

Optionsraum für das Integrationsdesign für die Synchronisierung von Kernsystemen

Topologie	Paradigma	Methode	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Kriterien Erfüllt	Bewertung
Bus	Storage-Driven	Database-Connection	3	3	3	3	2	4	3	7	7,33
Tree	Storage-Driven	Webhooks	3	3	4	3	2	3	3	7	7,64
Tree	Storage-Driven	Webservices	3	3	4	3	2	3	3	7	7,64
Tree	Storage-Driven	Change-Data-Capture	3	3	4	3	1	4	3	7	7,26
Tree	Storage-Driven	Database-Connection	3	3	4	3	1	4	3	7	7,45
Tree	Service-Driven	File-Imports	3	3	4	3	3	3	3	7	7,17
Hub	Storage-Driven	Webhooks	2	3	2	4	3	3	3	6	7,68
Hub	Storage-Driven	Webservices	2	3	2	4	3	3	3	6	7,68
Hub	Storage-Driven	Change-Data-Capture	2	3	2	3	2	4	3	6	7,30
Hub	Storage-Driven	Database-Connection	2	3	2	3	2	4	3	6	7,49
Hub	Service-Driven	File-Exports	1	4	2	3	3	3	3	6	7,20
Hub	Service-Driven	File-Imports	2	3	2	3	3	3	3	6	7,20
Hub	Service-Driven	Webhooks	1	4	2	4	4	2	4	6	7,82
Hub	Service-Driven	Webservices	1	4	2	4	4	2	4	6	7,82
Hub	Service-Driven	Change-Data-Capture	1	4	2	3	3	2	4	6	7,44
Hub	Service-Driven	Database-Connection	1	4	2	3	3	2	4	6	7,63
Hub	Event-Driven	File-Exports	1	4	2	3	3	3	3	6	7,20
Hub	Event-Driven	File-Imports	2	3	2	3	3	3	3	6	7,20
Hub	Event-Driven	Webhooks	1	4	2	4	4	2	4	6	7,82
Hub	Event-Driven	Webservices	1	4	2	4	4	2	4	6	7,82
Hub	Event-Driven	Change-Data-Capture	1	4	2	3	3	2	4	6	7,44
Hub	Event-Driven	Database-Connection	1	4	2	3	3	2	4	6	7,63
Bus	Storage-Driven	Webhooks	2	3	3	3	3	3	3	6	7,52
Bus	Storage-Driven	Webservices	2	3	3	3	3	3	3	6	7,52
Bus	Service-Driven	File-Exports	1	3	3	3	4	3	3	6	7,05
Bus	Service-Driven	Webhooks	1	3	3	3	4	2	4	6	7,67
Bus	Service-Driven	Webservices	1	3	3	3	4	2	4	6	7,67
Bus	Service-Driven	Change-Data-Capture	1	3	3	3	3	2	4	6	7,29
Bus	Service-Driven	Database-Connection	2	3	3	3	3	2	4	6	7,48
Bus	Event-Driven	Webhooks	2	3	3	3	4	3	4	6	7,67
Bus	Event-Driven	Webservices	2	3	3	3	4	3	4	6	7,67
Tree	Storage-Driven	File-Exports	3	3	4	3	2	4	2	6	7,02
Tree	Service-Driven	File-Exports	2	3	4	3	3	3	3	6	7,17
Tree	Service-Driven	Webhooks	2	4	4	3	3	2	4	6	7,79
Tree	Service-Driven	Webservices	2	4	4	3	3	2	4	6	7,79
Tree	Service-Driven	Change-Data-Capture	2	3	4	3	2	2	4	6	7,40

Topologie	Paradigma	Methode	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Kriterien Erfüllt	Bewertung
Tree	Service-Driven	Database-Connection	2	4	4	3	2	2	4	6	7,60
Tree	Event-Driven	Webhooks	2	3	4	3	3	3	4	6	7,79
Tree	Event-Driven	Webservices	2	3	4	3	3	3	4	6	7,79
Mesh	Storage-Driven	Webhooks	3	2	3	3	3	3	3	6	7,54
Mesh	Storage-Driven	Webservices	3	2	3	3	3	3	3	6	7,54
Mesh	Storage-Driven	Change-Data-Capture	3	2	3	3	2	4	3	6	7,15
Mesh	Service-Driven	Webhooks	2	3	3	3	4	2	4	6	7,68
Mesh	Service-Driven	Webservices	2	3	3	3	4	2	4	6	7,68
Mesh	Service-Driven	Change-Data-Capture	2	3	3	3	3	2	4	6	7,30
Mesh	Event-Driven	Database-Connection	3	3	3	2	3	3	4	6	7,49
Hub	Storage-Driven	File-Exports	2	3	2	3	2	4	2	5	7,06
Hub	Storage-Driven	File-Imports	3	2	2	3	2	4	1	5	7,06
Bus	Storage-Driven	File-Imports	3	2	3	3	3	4	1	5	6,90
Bus	Storage-Driven	Change-Data-Capture	2	2	3	3	2	4	3	5	7,14
Bus	Service-Driven	File-Imports	2	2	3	3	4	3	3	5	7,05
Bus	Event-Driven	File-Exports	2	3	3	2	4	4	3	5	7,05
Bus	Event-Driven	File-Imports	3	2	3	2	4	4	3	5	7,05
Bus	Event-Driven	Change-Data-Capture	2	3	3	2	3	3	4	5	7,29
Bus	Event-Driven	Database-Connection	2	3	3	2	3	3	4	5	7,48
Tree	Storage-Driven	File-Imports	4	2	4	3	2	4	1	5	7,02
Tree	Event-Driven	File-Exports	2	3	4	2	3	4	3	5	7,17
Tree	Event-Driven	File-Imports	3	2	4	2	3	4	3	5	7,17
Tree	Event-Driven	Change-Data-Capture	2	3	4	2	2	3	4	5	7,40
Tree	Event-Driven	Database-Connection	2	3	4	2	2	3	4	5	7,60
Mesh	Storage-Driven	Database-Connection	3	2	3	2	2	4	3	5	8,01
Mesh	Service-Driven	File-Exports	2	3	3	2	4	3	3	5	7,06
Mesh	Service-Driven	File-Imports	3	2	3	2	4	3	3	5	7,06
Mesh	Service-Driven	Database-Connection	2	3	3	2	3	2	4	5	7,49
Mesh	Event-Driven	File-Imports	3	2	3	2	4	4	3	5	7,06
Mesh	Event-Driven	Webhooks	2	3	3	2	4	3	4	5	7,68
Mesh	Event-Driven	Webservices	2	3	3	2	4	3	4	5	7,68
Bus	Storage-Driven	File-Exports	2	2	3	3	3	4	2	4	6,90
Mesh	Storage-Driven	File-Exports	3	2	3	2	3	4	2	4	6,92
Mesh	Storage-Driven	File-Imports	4	1	3	2	3	4	1	4	6,92
Mesh	Event-Driven	File-Exports	2	2	3	2	4	4	3	4	7,06
Mesh	Event-Driven	Change-Data-Capture	2	2	3	2	3	3	4	4	7,30

Ergebnisse des GAP-Modells für die Synchronisierung von Kernsystemen

	Robustheit	Integrität	Skalierbarkeit	Kontrolle	Flexibilität	Durchsatz	Latenz	Bewertung
Anforderungsprofil	3	3	1	3	1	1	3	
Option A: Storage-Driven Webservices in Tree	3	3	4	3	2	3	3	7,64
Option B: Storage-Driven Database-Connection in Tree	3	3	4	3	1	4	3	7,45
Option C: Storage-Driven Database-Connection in Bus	3	3	3	3	2	4	3	7,33
Implementierung 1: Storage-Driven File-Exports in Hub	2	3	2	3	2	4	2	7,06
Implementierung 2: Storage-Driven Webservices in Hub	2	3	2	4	3	3	3	7,68
Implementierung 3: Event-Driven Webservices in Hub	1	4	2	4	4	2	4	7,82

Ergebnisse des GAP-Modells in Fallstudie 3

Anhang 15 Von KI generierter Code Systemintegration

Default (GPT-3.5)

K Create python code that extracts data from a system via webservices and copies this into another system via ODBC every five minutes. The Webservice has a dedicated endpoint to provide a delta-log

To create Python code that extracts data from a system via web services and copies it into another system via ODBC every five minutes, you can use the following approach. You will need to install the necessary libraries if you haven't already. You can use `requests` for web services and `pyodbc` for ODBC connectivity. Additionally, you can use a scheduler like `schedule` to run the job at regular intervals.

Here's a basic outline of the code:

```
python Copy code

import requests
import pyodbc
import schedule
import time

# Configuration for web service and ODBC
web_service_url = "https://example.com/web_service_endpoint" # Replace with
odbc_connection_string = "DRIVER={ODBC Driver};SERVER=your_server;DATABASE=y

# Define a function to extract data from the web service
def extract_data_from_web_service():
    try:
        # Make a request to the web service endpoint for delta-log data
        response = requests.get(web_service_url)
        if response.status_code == 200:
            data = response.json() # Adjust this depending on the response
            return data
        else:
            print(f"Failed to retrieve data from the web service. Status Code")
            return None
    except Exception as e:
        print(f"Error while fetching data from the web service: {str(e)}")
        return None

# Define a function to copy data to the ODBC database
def copy_data_to_odbc(data):
    try:
        # Establish an ODBC connection
        conn = pyodbc.connect(odbc_connection_string)
        cursor = conn.cursor()
```

```
# Assuming 'data' is a list of records, iterate and insert into the
for record in data:
    # Replace with your own SQL INSERT statement
    sql = f"INSERT INTO your_table (column1, column2, ...) VALUES ({
    cursor.execute(sql, tuple(record.values()))

# Commit the changes and close the connection
conn.commit()
conn.close()
print("Data copied to ODBC successfully.")
except Exception as e:
    print(f"Error while copying data to ODBC: {str(e)}")

# Schedule the job to run every 5 minutes
schedule.every(5).minutes.do(lambda: copy_data_to_odbc(extract_data_from_web

# Main loop to run the scheduled tasks
while True:
    schedule.run_pending()
    time.sleep(1) # Sleep for 1 second to avoid high CPU usage
```

Make sure to replace the placeholders with your actual web service URL, ODBC connection string, and database table structure. This code will fetch data from the web service, process it, and insert it into the ODBC database at five-minute intervals. You may need to adjust the data extraction and insertion logic to match your specific requirements and data format.

Von KI generierter Code für eine Storage-Driven Integration mit Webservices und Database-Connection

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 07.02.2025

Konrad Nadobny