Der kostenoptimale Grad funktionaler Redundanz in IT-Anwendungslandschaften

Dr. Thomas Widjaja

Technische Universität Darmstadt
Hochschulstraße 1
64289 Darmstadt
widjaja@is.tu-darmstadt.de

Abstract: Für die Entstehung von unerwünschter funktionaler Redundanz in IT-Anwendungslandschaften existieren zahlreiche Gründe: So können beispielsweise dezentrale Entscheidungsstrukturen oder Unternehmenszusammenschlüsse dazu führen, dass unterschiedliche IT-Anwendungen eine identische fachliche Funktion anbieten. Anders als bei geplanter funktionaler Redundanz – wie beispielsweise bei einem Backup-System – verursacht diese unerwünschte Redundanz Mehrkosten ohne Nutzen zu stiften. Die Reduktion dieser Redundanzen ist jedoch auch mit Kosten für die Konsolidierung der redundanten Anwendungen verbunden. In diesem Artikel wird ein mathematisches Modell vorgestellt, dass IT-Architekten dabei unterstützt, den kostenoptimalen Grad an funktionaler Redundanz in der IT-Anwendungslandschaft zu bestimmen. Das Modell wird im Rahmen eines realen Anwendungsbeispiels evaluiert und dient als Grundlage für praxisrelevante Handlungsempfehlungen und theoretische Implikationen.

1 Einleitung

IT-Anwendungslandschaften sind komplexe Systeme: Eine Vielzahl von Komponenten ist durch unterschiedliche Relationen (Schnittstellen) untereinander verbunden. In einem solchen System existieren oftmals funktionale Redundanzen, d. h. eine identische fachliche Aufgabe wird durch mehr als eine Komponente abgebildet. Diese funktionale Redundanz ist beispielsweise die Folge von unkontrolliertem und unkoordiniertem historischen Wachstum der IT-Landschaft oder entsteht durch externe Einflüsse wie Unternehmenszusammenschlüsse.

Aus der Literatur ist bekannt, dass unnötige funktionale Redundanz[1] ein wesentlicher Treiber für die Betriebskosten der IT-Landschaft ist und, dass dieses Problem durch Standardisierungsmaßnahmen angegangen werden kann. [BY06] Einige Quellen legen sogar den Schluss nahe, dass in der Erkennung und dem Management dieser unerwünschten funktionalen Redundanzen ein wesentlicher Wert des kompletten Enterprise Architecture Managments (EAM) liegt. [z. B. EBD11, Ta11, Za87] In der Praxis zeigt

[1] In diesem Artikel wird ausschließlich die unerwünschte funktionale Redundanz behandelt. Dabei bleibt unberührt, dass es auch gewünschte funktionale Redundanz innerhalb einer IT-Landschaft geben kann. Beispiele für solche gewollte Redundanz sind Dopplungen von Komponenten, um die Ausfallsicherheit zu erhöhen.

sich jedoch, dass die vollständige Reduktion der identifizierten unnötigen Redundanz weder sinnvoll noch möglich ist. Ein Grund dafür ist, dass die Reduktion von bereits existierender funktionaler Redundanz mit erheblichen Kosten verbunden ist. Da Kostenmanagement eine wesentliche Anforderung an CIOs und IT-Architekten ist [GP12, Li06], stellt sich insbesondere für sie die Frage, wie mit existierender Redundanz in den IT-Landschaften umzugehen ist und entsprechend, wie hoch der kostenoptimale Grad an Redundanz in der Zielarchitektur [Ai09] sein sollte.

In der aktuellen Literatur fehlt zum einen die detaillierte Konzeptualisierung des zugrundeliegenden Trade-offs und zum anderen wird bisher kein Entscheidungsunterstützungsmodell vorgeschlagen, das die gegenläufigen Kosteneffekte (insb. „unnötige" Betriebskosten redundanter Anwendungen vs. Kosten der Redundanzreduktion - siehe auch Abbildung 1) zu einer Lösung integriert. Bisher ist es oftmals eine in der Praxis akzeptierte Lösung, in der Ziel-Architektur [Ai09] die identifizierten unerwünschten funktionalen Redundanzen so weit wie möglich zu reduzieren – bis zu welchem Grad dies jedoch ökonomisch sinnvoll ist, bleibt dem „Bauchgefühl" der IT-Architekten überlassen. Die in diesem Artikel vorgeschlagene Konzeptualisierung des zugrundeliegenden Trade-Offs ist damit sowohl für Praxis als auch Wissenschaft relevant, da zum einen ein wesentlicher Mechanismus der Wertgenerierung von EAM detaillierter beleuchtet wird und zum anderen die Grundlagen für ein praxisrelevantes Entscheidungsunterstützungssystem erarbeitet werden.

In Summe zielt der vorliegende Artikel darauf ab, folgende Forschungsfrage zu beantworten: Wie können IT-Architekten bei der Ermittlung des kostenoptimalen Grads an funktionaler Redundanz in einer Ziel-Architektur unter Berücksichtigung der gegenläufigen Kosteneffekte unterstützt werden? Der Artikel ist wie folgt strukturiert: Im nächsten Kapitel wird das beschriebene Problem theoretisch eingeordnet und die bereits existierende Forschung zum Thema aufgearbeitet. In Kapitel 3 wird die zugrundeliegende Problemklasse definiert. Auf dieser Basis wird in Kapitel 4 eine Formalisierung des grundlegenden Trade-offs als Optimierungsproblem vorgeschlagen. Die vorgestellte Modellformulierung wird in Kapitel 5 im Rahmen eines realen Anwendungsfalls in Kooperation mit einem Praxisunternehmen evaluiert. Nachdem die praktische Nutzbarkeit geprüft ist, werden anschließend in Kapitel 6 die Implikationen sowohl aus praktischer als auch theoretischer Sicht diskutiert.

2 Theoretischer Hintergrund

2.1 IT-Architektur aus Systemtheoretischer Perspektive

In diesem Artikel wird [IEEE00] folgend, eine IT-Architektur als System verstanden, das sich aus Komponenten und deren Relationen untereinander zusammensetzt. Diese Sichtweise entspricht damit der klassischen Systemtheorie [HF69]. Dieses hierarchische System [Si62] kann in verschiedene untereinander verbundene Sub-Systeme untergliedert werden: Im EAM-Kontext werden insbesondere Anwendungs-Architektur, Daten-Architektur und Infrastruktur-Architektur unterschieden (z. B. [RWR06]). Im Folgenden

fokussiert der Artikel auf die Anwendungs-Architektur eines Unternehmens. Damit sind die Komponenten des betrachteten Systems IT-Anwendungen und die entsprechenden Relationen repräsentieren Schnittstellen zum Datenaustausch.

2.2 Der Redundanzbegriff im Kontext von IT-Architekturen

Redundanz, von lateinisch redundantia „Überfülle“, beschreibt das Vorhandensein von eigentlich überflüssigen, identischen oder vergleichbaren Ressourcen. Herbert Simon definiert „If a complex structure is completly **un**redundant [Markierung hinzugefügt]- if no aspect of its structure can be inferred from any other - then it is its own simplest description.“ [Si96, S. 209] Im Kontext von IT-Architekturen wird die Förderung von Wiederverwendung und damit die Vermeidung von Redundanzen in zahlreichen Quellen als einer der wesentlichen Nutzen der EAM-Funktion gesehen. [z. B. EBD11, Ta11, Za87] Im Zusammenhang von IT-Systemen sind insbesondere die in diesem Artikel betrachtete funktionale Redundanz und die Datenredundanz von Bedeutung. Datenredundanz bezeichnet eine Situation, in der ein und dieselbe Information an mehr als einem Ort gespeichert ist. Funktionale Redundanz liegt vor, wenn mehr als eine Komponente innerhalb des Systems (siehe Kapitel 2.1) die gleiche Funktion erfüllt. Da die in diesem Artikel betrachteten Komponenten des Systems IT-Anwendungen sind, kann der Redundanzbegriff konkretisiert werden: Da die Kernaufgabe einer IT-Anwendung die Informationstransformation ist, sind im Sinne dieses Artikels zwei Anwendungen funktional redundant, wenn sie für alle Eingabe-Informationen die gleiche Ausgabe-Informationen liefern. Diese Definition von Redundanz ist angelehnt an die Definition von [BB03] aus dem Prozesskontext: „redundancy-that is, a determination of whether there is more than one way to connect an input to an output.“ (S. 339) Dieses Redundanzverständnis lässt neben der beschriebenen binären Klassifizierung, d. h. „vollständiger Redundanz“ (jede Eingabe-Information wird zu einer identischen Ausgabe-Information transformiert), auch eine graduelle Variante zu, d. h. X % der Eingaben führen zu identischen Ausgaben. Zur Vereinfachung wird die folgende Betrachtung, auf das binäre Verständnis fokussiert – die entsprechenden Limitationen dieses Ansatzes werden jedoch in Kapitel 6 wieder aufgegriffen. Dieses Redundanzverständnis ist nicht kommutativ, d. h. aus „Anwendung A ist redundant zu Anwendung B“ folgt *nicht*, dass auch B redundant zu A ist. So kann beispielsweise die vollständige Funktion einer Anwendung A gleichzeitig von Anwendung B angeboten werden - Anwendung B jedoch darüber hinaus wesentlich mehr Funktionen zur Verfügung stellen. Daher kann zwar A durch B aber nicht B durch A ersetzt werden. Der in diesem Artikel verwendete Redundanzbegriff ist jedoch transitiv - denn aus „A ist redundant zu B“ und „B ist redundant zu C“ folgt, dass A redundant zu C ist.

Im Kontext von IT-Architekturen kann erforderliche und unnötige funktionale Redundanz unterschieden werden. Beispiele für erforderliche Redundanz – deren Ziel meist die Erhöhung der Ausfallsicherheit des Systems ist – sind Backup-Systeme, die bei Versagen der Hauptkomponenten die wesentlichen Funktionalitäten übernehmen. Unnötige funktionale Redundanz – also Redundanz bei der die entstehenden Kosten keinem angemessenen Nutzen gegenüberstehen – kann beispielsweise durch unkoordiniertes Wachstum der IT-Landschaft oder Unternehmenszusammenschlüsse entstehen. [z. B.

LC07] Im Folgenden wird auf diese unnötige funktionale Redundanz als Ziel möglicher Konsolidierungsmaßnahmen fokussiert.

2.3 Reduktion von unerwünschter funktionaler Redundanz

Während Standards die „Entstehung“ von Redundanz verhindern können [BY06], zielt Konsolidierung auf die Reduktion von existierender Redundanz ab. Der Begriff „Konsolidierung“ bezeichnet das Zusammenfassen von Elementen zu einem Ganzen. Entsprechend wird in diesem Artikel unter (Anwendungs-)Konsolidierung das Zusammenfassen von zwei funktional redundanten Anwendungen verstanden. Hierbei ist das Ziel, eine der beiden Anwendungen nicht weiter zu betreiben. Dieses Verständnis des Konsolidierungsbegriffes wird im Kontext des IT-Architekturmanagements als Sonderform der IT-Integration verstanden (siehe z. B. [KWS05]). In der Terminologie von [LC07] entspricht der oben beschriebene Konsolidierungsbegriff der „Choose one“-Integrationsstrategie und [BB03] beschreiben dieses Vorgehen als „Synthesis“-Ansatz.

[CLT12] leiten aus ihrem aktuellen und umfassenden Literaturüberblick ab, dass aktuell eine Theorie der “Integration” in der Wirtschaftsinformatik fehlt. Ähnlich argumentieren [LC07], die in einer Serie von Fallstudien Industrieerfahrungen zum Management von funktionaler Redundanz von Software aggregieren. Da im vorliegenden Artikel ein zentraler Trade-off bei der Konsolidierung von IT-Anwendungen (d. h. einem wesentlichen Teilbereich der IT-Integration) konzeptualisiert und formalisiert wird, bietet der vorliegende Artikel einen ersten Ansatzpunkt in diese Richtung.

3 Die untersuchte Problemklasse: Ermittlung des kostenoptimalen Grades von unerwünschter Redundanz in einer IT-Anwendungslandschaft

Existiert in einer IT-Landschaft unerwünschte funktionale Redundanz, gilt es zu prüfen bis zu welchem Grad es ökonomisch sinnvoll ist, diese zu reduzieren. Die Bestimmung des kostenoptimalen Grades von unnötiger funktionaler Redundanz unterliegt folgendem fundamentalen Trade-Off (siehe Abbildung 1): Auf der einen Seite verursachen die „unnötigen“ Anwendungen und die entsprechenden notwendigen Schnittstellen zusätzliche Betriebskosten und Kommunikationskosten.[2] Auf der anderen Seite ist die Reduktion dieser unerwünschten funktionalen Redundanz mit Konsolidierungskosten[3] und ggf. den Kosten für das Entwickeln neuer Schnittstellen und den entsprechenden Kommunikati-

[2] Betriebskosten sind laufende Kosten und umfassen nach der Klassifikation von [Bu01, S. 26ff] folgende Oberkategorien: Systemkosten, Personalkosten, Fremdleistungskosten und sonstige Kosten. Kommunikationskosten sind in Anlehnung an [Wi11] als laufende Kosten für die Informationsübermittlung zwischen zwei Anwendungen definiert.

[3] Konsolidierungskosten sind einmalige Kosten und sind in Anlehnung an [Bu01S. 26ff] als Anschaffungs- und Herstellkosten, Personalkosten, Fremdleistungskosten, Installations- und Implementierungskosten und sonstige Kosten für die Überführung einer redundanten Anwendung A in Anwendung B sowie die Abschaltung von Anwendung A definiert. Siehe [Bu01S. 26ff] für eine detaillierte Aufschlüsselung der aufgelisteten Oberkategorien. Analog sind auch die einmaligen Entwicklungskosten für neue Schnittstellen definiert.

onskosten verbunden. Das Entwickeln neuer Schnittstellen kann nötig sein, da es nach der Konsolidierung der redundanten Anwendung A in die Anwendung B zu Veränderungen des Informationsbedarfs von Anwendung B kommen kann (da Anwendung B in der Zielarchitektur auch mit den ehemaligen Kommunikationspartnern von A Informationen austauschen muss).

Da es sich bei den betrachteten Größen sowohl um laufende Kosten (z. B. Betriebskosten für Anwendungen) als auch um einmalige Kosten (z. B. Kosten für die Konsolidierung der entsprechenden Anwendungen) handelt, ist es notwendig die laufenden Kosten über einen definierten Betrachtungszeitraum zu diskontieren.

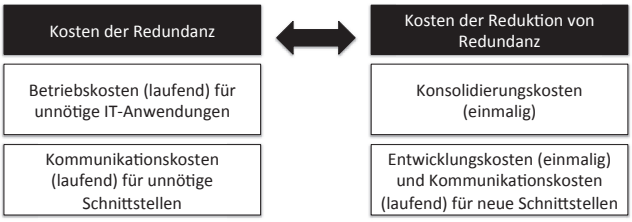

Abbildung 1: Zugrundeliegender Trade-Off

Die Schwierigkeit den kostenoptimalen Konsolidierungsplan zu identifizieren, steigt mit der Komplexität der Redundanz- und den Kommunikationsbeziehungen zwischen den Anwendungen. Eine hohe Komplexität dieser Beziehungen ist beispielsweise zu erwarten, wenn nach einem Merger zweier Finanzdienstleister zwei unterschiedliche technische Umsetzungen zur Unterstützung eines Kreditvergabeprozess existieren. Ein solcher Ausschnitt aus der IT-Anwendungslandschaft ist stark vereinfacht auf der linken Seite von Abbildung 2 als dreigliedriger Prozess (Management der Kundenanfrage, Bonitätsprüfung, Verbuchung) dargestellt. Durch die gerichtete Modellierung der Redundanzbeziehung (gestrichelte Pfeile in Abbildung 2) kann folgende Situation berücksichtigt werden: Während Anwendung A (zum Beispiel das Management der Kundenanfragen in Unternehmen 1) redundant zu Anwendung A' und vice versa ist, gilt dies nicht für Anwendung B und Anwendung B' (die in diesem Beispiel beide die Bonitätsprüfung unterstützen). Zwar ist Anwendung B redundant zu Anwendung B', jedoch nicht umgekehrt (beispielsweise könnten in Anwendung B' wesentlich umfassendere Prüfalgorithmen implementiert sein). Dieses stark vereinfachte Beispiel verdeutlicht zudem, dass eine mögliche Konsolidierung von redundanten Anwendungen neben der Reduktion der Anzahl von Anwendungen auch zur Veränderung der Kommunikationsbeziehungen (ungerichtete durchgezogene Kanten in Abbildung 2) zwischen den Anwendungen führen kann: Für den Fall, dass in der Zielarchitektur Anwendung A' in Anwendung A konsolidiert wird (wie in der Lösung I in Abbildung 2 dargestellt), fallen zum einen Kosten für die Konsolidierung (Zusammenfassen von A und A') und zum anderen Entwicklungs- und Kommunikationskosten für eine neue Schnittstelle zwischen Anwendung A und Anwendung B' an. Demgegenüber können die Betriebskosten für Anwendung A' sowie die Kommunikationskosten für die Schnittstelle zwischen A' und B' eingespart werden. Eine alternative Lösung könnte darin bestehen, die IT-Anwendungen von Unternehmen 1 soweit wie möglich in die IT-Landschaft von Unternehmen 2 aufgehen zu lassen, wie in der Lösung II in Abbildung 2 dargestellt.

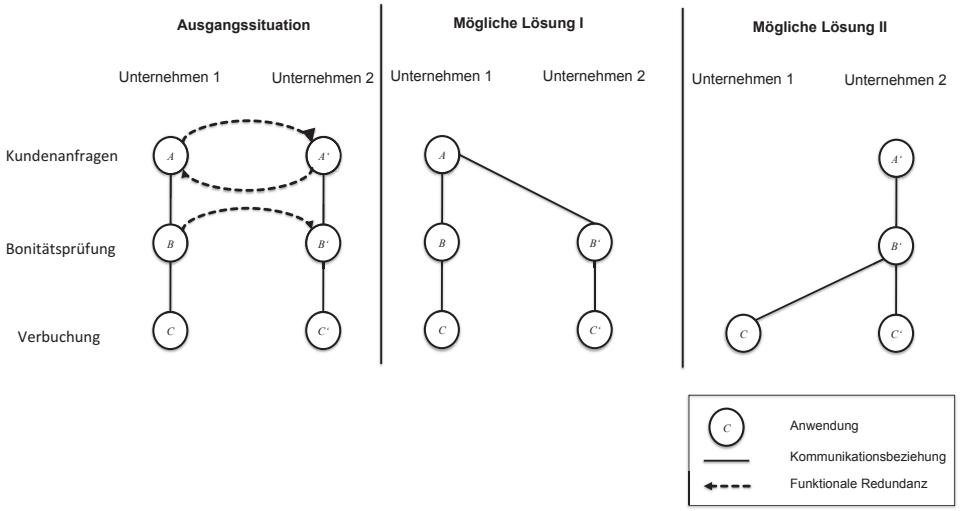

Abbildung 2: Beispielhafte Darstellung der Anwendungslandschaft nach dem Merger von zwei Banken

IT-Architekten wurden bisher bei dem komplexen Entscheidungsproblem, den optimalen Konsolidierungsplan zu identifizieren, nicht durch ein theoretisch fundiertes und formalisiertes Modell unterstützt. Diese Lücke wird im folgenden Kapitel geschlossen.

4 Bestimmung des kostenoptimalen Grads an funktionaler Redundanz als Optimierungsproblem

Zur Formalisierung des Entscheidungsproblems wird die IT-Architektur (d. h., das betrachtete System[4]) als Graph modelliert (siehe Abbildung 3). Die Konten $V = \{1,2,\dots,n\}$ entsprechen dabei den IT-Anwendungen und es werden zwei Arten von Relationen zwischen den Anwendungen unterschieden: Kommunikationsbeziehungen $E_{Com} = V \times V$ (ausgefüllte Kanten) und Redundanzbeziehungen $E_{Red} = V \times V$ (gestrichelte Pfeile).

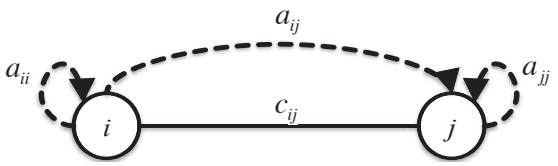

Abbildung 3: Dem Modell zugrundeliegender Graph

[4] Hierbei ist zu beachten, dass die Anwendungslandschaft als System bezüglich verschiedener Abstraktionsebenen betrachtet werden kann [Si62]: Eine Kernbankenanwendung kann zum Beispiel auch selbst als System verstanden werden, das aus unterschiedlichen Sub-Systemen (Modulen) besteht. Für die sinnvolle Anwendung des Modells ist durch den IT-Architekten eine geeignete Abstraktionsebene zu definieren.

Redundanzbeziehungen geben an, dass zwei Anwendungen zusammengefasst werden können und sind mit den Kosten für die Konsolidierung $a_{ij} \in \mathbb{R}_+ \forall (i,j) \in E_{Red}$ gewichtet (siehe Abbildung 3). Als Sonderfall – um eine einfache Modellierung zu ermöglichen - existieren „virtuelle" funktionale Redundanzen, die von jedem Knoten auf sich selbst gerichtet sind (sog. Schlingen) und mit den jeweiligen diskontierten Betriebskosten der Anwendung gewichtet werden.[5] Der Parameterwert $a_{ij} \in \mathbb{R}_+$ repräsentiert also entweder die Betriebskosten der Anwendung *i* (falls $i = j$) oder die Konsolidierungskosten (falls $i = j$).

$$a_{ij} = \begin{cases} Betriebskosten\ von\ Anwendung\ i & | i = j \\ Konsolidierungskosten & | i\ redundant\ zu\ j\ und\ i = j \end{cases} \quad [1]$$

Die ungerichteten Kommunikationsbeziehungen zwischen den Anwendungen repräsentieren Schnittstellen und sind mit diskontierten Kommunikationskosten $c_{ij} \in \mathbb{R}_+ \forall i,j \in V\ und\ i < j$ gewichtet.[6] Zur einfacheren Modellierung wird zudem die Menge $Com_i = \{j | (i,j)\ oder\ (j,i) \in E_{Com}\} \forall i \in V$ definiert, die alle direkten Kommunikationspartner von Anwendung *i* enthält. Zudem ist es möglich, dass Anwendung *i* im Rahmen von Konsolidierungsmaßnahmen neue Kommunikationspartner erhält und entsprechend neue Schnittstellen entwickelt werden müssen – in diesen Fällen müssen neben den Kommunikationskosten für die neue Schnittstelle auch einmalige Entwicklungskosten berücksichtigt werden (in Gleichung [2] als „$Entw. = und\ Kom. = kosten$" bezeichnet). Die Menge $PotCom_i$ enthält alle potenziellen Kommunikationspartner von $i \in V$ in einer Zielarchitektur. Es werden auch diejenigen Anwendungen berücksichtigt, die mit einem aktuellen Kommunikationspartner funktional redundant sind bzw. deren Kommunikationspartner. Damit kann $c_{ij} \in \mathbb{R}_+$ definiert werden als:

$$c_{ij} = \begin{cases} Kommunikationskosten\ zw.\ i\ und\ j & | j \in Com_i\ und\ i < j \\ Entw. = und\ Kom. = kosten\ zw.\ i\ und\ j & | j \in PotCom_i \setminus Com_i\ und\ i < j \\ 2 & | sonst \end{cases} \quad [2]$$

Zur Formulierung des Optimierungsproblems sind zwei binäre Entscheidungsvariablen notwendig. Die Variable $x_{ij} \in \{2,1\}\ \forall (i,j) \in E_{Red}$ nimmt genau dann den Wert 1 an, wenn in der Zielarchitektur die Anwendung *i* in Anwendung *j* konsolidiert wird. Es ist explizit möglich, dass $i = j$ gilt – genau dann, wenn Anwendung *i* „weiterbetrieben" wird.

$$x_{ij} = \begin{cases} 1 & | Anwendung\ i\ in\ Anwendung\ j\ konsolidiert \\ 2 & | sonst \end{cases} \quad [3]$$

Die zweite Entscheidungsvariable $y_{ij} \in \{2,1\}\ \forall i,j \in V$ nimmt genau dann den Wert 1 an, wenn in der Zielarchitektur eine Kommunikationsbeziehung zwischen Anwendung *i* und Anwendung *j* existiert.

[5] Konzeptionell besteht zwischen *i* und *i* also eine funktionale Redundanz.

[6] Da die Kommunikationsbeziehungen ungerichtet sind, ist es ausreichend nur eine Hälfte der Matrix von $c_{ij} \in \mathbb{R}_+$ zu spezifizieren (also für $i < j$).

$$y_{ij} = \begin{cases} 1 & |Anwendung\ i\ tauscht\ mit\ Anwendung\ j\ Informationen\ aus \\ 2 & |sonst \end{cases} \quad [4]$$

In Tabelle 1 sind die verwendeten Symbole sowie deren Bedeutung zusammengefasst.

Tabelle 1: Verwendete Symbole

Symbol	Bedeutung
$\boldsymbol{V = \{1, 2, \dots, n\}}$	Anwendungen
$\boldsymbol{E_{Com} = V \times V}$	Kommunikationsbeziehungen
$\boldsymbol{Com_i = \{j \mid (i,j) oder\ (j,i) \in E_{Com}\} \forall i \in V}$	Kommunikationspartner von *i*
$\boldsymbol{PotCom_i = V}$	Mögliche Kommunikationspartner von *i* in der Zielarchitektur
$\boldsymbol{E_{Red} = V \times V}$	Redundanzbeziehungen
$\boldsymbol{a_{ij} \in \mathbb{R}_+ \forall (i,j) \in E_{Red}}$	Konsolidierungs- bzw. Betriebskosten
$\boldsymbol{c_{ij} \in \mathbb{R}_+ \forall i,j \in V, i < j}$	Entwicklungs- + Kommunikationskosten
$\boldsymbol{x_{ij} \in \{0,1\} \forall (i,j) \in E_{Red}}$	Entscheidungsvariable, die genau dann den Wert 1 annimmt, wenn *i* in *j* betrieben wird. Hinweis: $x_{ii} = 1$ entspricht dem Weiterbetrieb von *i*.
$\boldsymbol{y_{ij} \in \{0,1\}\ \forall i,j \in V}$	Entscheidungsvariable, die genau dann den Wert 1 annimmt, wenn *i* mit *j* Informationen austauscht.
$\boldsymbol{F(x,y) \in \mathbb{R}_+}$	Zielfunktion (Gesamtkosten)
$\boldsymbol{i, i', j, k, m, l \in V}$	Indizes (die jeweils Anwendungen repräsentieren)

Mithilfe dieser Notation kann folgendes Optimierungsproblem formuliert werden:

$$Min!\ F(\boldsymbol{x},\boldsymbol{y}) \coloneqq \underbrace{\sum_{\substack{i,j\ mit \\ (i,j) \in E_{Red}}} x_{ij} \cdot a_{ij}}_{\substack{Konsolidierungs+ \\ und\ Betriebskosten}} = \underbrace{\sum_{\substack{i,j \in V \\ i<j}} y_{ij} \cdot c_{ij}}_{\substack{Entwicklungs+und \\ Kommunikations+ \\ kosten}} \quad [5]$$

Unter den Nebenbedingungen

$$\sum_{\substack{j\ mit \\ (i,j) \in E_{Red}}} x_{ij} = 1 | \forall\ i \in V \quad [6]$$

$$x_{ij} < x_{jj} | \forall\ (i,j) \in E_{Red} \quad [7]$$

$$x_{ij} < y_{jk} | \forall\ (i,j) \in E_{Red}\ und\ k \in \{l | x_{ml} = 1 \wedge m \in Com_i\} \quad [8]$$

$$y_{ij} = y_{ij} | \forall\ i,j \in V \quad [9]$$

$$x_{ij}, y_{kl} \in \{2,1\} | \forall\ (i,j) \in E_{Red}\ und\ k,l \in V \quad [10]$$

Die Zielfunktion berücksichtigt Betriebskosten (Summe 1 für $i = j$), Konsolidierungskosten (Summe 1 für $i = j$) sowie Entwicklungs- und Kommunikationskosten (Summe 2) in der Zielarchitektur. Die erste Nebenbedingungsgruppe (Gleichung 6) garantiert,

dass jede Anwendung entweder weiterbetrieben wird ($i = j$) oder mit einer anderen j Anwendung (zu der i funktional redundant ist) konsolidiert wird. Die zweite Gruppe von Nebenbedingung (Gleichung 7) garantiert, dass sobald eine Anwendung j das „Ziel" einer Konsolidierung ist (d. h. es existiert ein $x_{ij} = 1$), Anwendung j tatsächlich betrieben wird. Die dritte Gruppe von Nebenbedingungen (Gleichung 8) garantiert, dass die Anwendung, die Ziel der Migration ist, alle Kommunikationsbeziehungen der migrierten Anwendung „erbt". Wird also Anwendung i mit Anwendung j konsolidiert, (d. h. $x_{ij} = 1$) muss gelten $y_{jk} = 1$ (wobei der Index k Anwendungen beschreibt, zu denen ehemalige Kommunikationspartner von i (d. h. $m \in Com_i$) konsolidiert wurden). Hierbei zeigt sich u. a. auch, warum die Modellierung der Betriebskosten durch Schlingen im Graph (z. B. $x_{ii} = 1$) nützlich ist. Denn für die Variablen $x_{ml} = 1$ in Gleichung 8 kann somit gelten $m = l$ (genau dann, wenn die Kommunikationspartner von i nicht in anderen Anwendungen konsolidiert wurden). Für den Fall $i = j$ (d. h. die Anwendung i selbst wird weiterbetrieben) garantiert Gleichung 8 zudem, dass $y_{ik} = 1$ gilt, also Anwendung i weiterhin mit den bisherigen Kommunikationspartnern oder deren Konsolidierungszielen kommuniziert.[7] Die vierte Gruppe von Nebenbedingungen (Gleichung 9) ist notwendig, da nur ungerichtete Kommunikationsbeziehungen für $i < j$ betrachtet werden, die Entscheidungsvariable jedoch für $\forall\, i, j \in V$ definiert ist (Gleichung 10). So ist es möglich, die Formulierung der vierten Nebenbedingungsgruppe (Gleichung 8) zu vereinfachen. Die fünfte Gruppe von Nebenbedingungen (Gleichung 10) definiert die binären Wertebereiche der Entscheidungsvariablen.

5 Anwendung des Modells bei einem Dienstleistungsunternehmen

Das in Kapitel 3 konzeptualisierte und in Kapitel 4 formalisierte Entscheidungsproblem wurde für ein internationales Dienstleistungsunternehmen mit mehr als 100.000 Mitarbeitern und für einen Teil der dort historisch gewachsenen IT-Landschaft instanziiert. Insgesamt betreibt das Unternehmen mehr als 200 Anwendungen. Gemeinsam mit zwei Experten wurde der in diesem Artikel präsentierte illustrative Ausschnitt der IT-Architektur ausgewählt. Hierbei wurden 21 Anwendungen, mit 19 Schnittstellen und 11 entsprechende Konsolidierungsmaßnahmen (die in acht Konsolidierungsprojekten gruppiert sind: P01a bis P08)[8] ausgewählt. Der betrachtete Ausschnitt der IT-Landschaft umfasst die wesentlichen IT-Systeme für den Vertrieb des Unternehmens und umfasst mehrere Vertriebskanäle (Filiale, Internet, Großkunden) von mehreren Geschäftsdivisionen inkl. der angrenzenden Hilfssysteme wie Stammdatenmanagement, Kundenbetreuung, Reklamationsmanagement und Customer-Relationship-Management. Ebenfalls betrachtet wurden die Planungssysteme für Personal, Finanzen und Produktion, die in direkter Abhängigkeit der vertriebsunterstützenden Geschäftsprozesse stehen, mitberücksichtigt.

[7] Generell gilt für alle $y_{ij} = 2 | \forall\, i, j \in V$ die nicht in Gleichung 8 oder Gleichung 9 auf den Wert 1 „gezwungen" werden, da $c_{ij} \in \mathbb{R}_+ \forall i, j \in V$ nur positive Werte annehmen kann.

[8] Aufgrund einer besonderen Anforderung in diesem Anwendungsfall wurde über zusätzliche Nebenbedingungen sichergestellt, dass Konsolidierungsprojekte mit einer identischen Projektnummer nur gemeinsam durchgeführt werden (daher die verwendete Nummerierung). Beispielsweise kann „P01a" nur durchgeführt werden, wenn auch „P01b" durchgeführt wird (und umgekehrt).

Für diesen Ausschnitt der IT-Landschaft sind vor der Konsolidierung 32 Mio. € Betriebskosten und 5 Mio. € Kommunikationskosten angefallen. Anschließend wurden auf Grundlage von existierenden Dokumenten zur Kostenschätzungen und in enger Zusammenarbeit mit zwei Experten und einem weiteren Wissenschaftler die für die Modellinstanziierung notwendigen Parameterwerte geschätzt.[9]

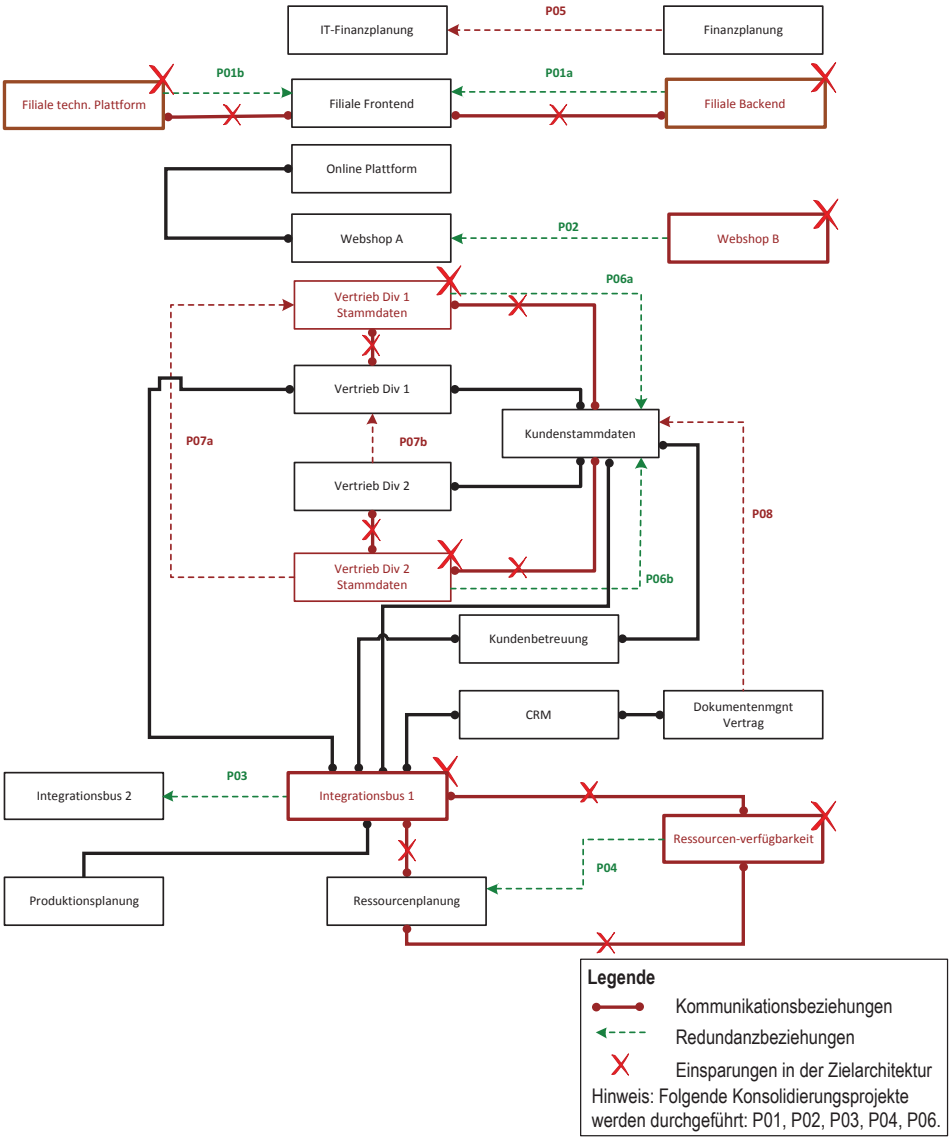

Abbildung 4: Betrachteter Ausschnitt der IT-Landschaft und Darstellung der Zielarchitektur (für eine Darstellung der Soll-Architektur siehe Abbildung 5 im Anhang)

[9] Die ermittelten Parameterwerte werden auf Anfrage an widjaja@is.tu-darmstadt.de zur Verfügung gestellt.

Nach der Modellanwendung umfasst der kostenoptimale Konsolidierungsplan sieben (in grün dargestellt) der elf möglichen Konsolidierungsmaßnahmen. Insgesamt können so sieben funktional redundante Anwendungen (in rot dargestellt) und neun entsprechende Kommunikationsbeziehungen (in rot dargestellt) eingespart werden. Fünf Kommunikationsbeziehungen bleiben unberührt und sechs weitere Kommunikationsbeziehungen müssen aufgrund von Konsolidierungsprojekten neu entwickelt werden. Damit konnten die Gesamtkosten von ca. 37 Mio. € (32 Mio. € Betriebskosten und 5 Mio. € Kommunikationskosten) um ca. 23 % auf ca. 28,5 Mio. € reduziert werden. Während der Evaluation standen insbesondere die Kriterien Validität und Nützlichkeit im Vordergrund. [GH13] Die Ergebnisse des Optimierungsmodells stimmten mit den Erfahrungen der Experten überein und sie betonten die Nützlichkeit des Modells für vergleichbare Konsolidierungsprojekte. Daneben zeigte die Evaluation zudem interessante Ansatzpunkte für weitere Forschung, die im nächsten Kapitel diskutiert werden.

6 Diskussion

Theoretische Implikationen

Die zwei wesentlichen Ergebnisse dieses Artikels sind: 1) Identifikation und Konzeptualisierung des beschriebenen Trade-offs zur Ermittlung des kostenoptimalen Grads an unerwünschter funktionaler Redundanz in einer IT-Architektur. 2) Formalisierung und Lösung dieses Trade-offs mithilfe eines mathematischen Modells. Die hier erarbeiteten Ergebnisse können in die Literatur zu EAM einfließen, da die Identifikation und das Management funktionaler Redundanzen einen großen Wert für Unternehmen stiften kann. [RWR06, Ta11] Nach [LC07], die in einer Serie von Fallstudien Industrieerfahrungen zum Management von funktionaler Redundanz von Software aggregieren, gilt jedoch: „There is no established theory or methods that specifically address this situation." (p. 420) Daher ist die Identifikation und Formalisierung des beschriebenen Trade-offs ein wichtiger erster Schritt in diese Richtung und unterstützt CIOs und IT-Architekten beim Kostenmanagement. [GP12, Li06] Zudem kann die Identifikation des in diesem Artikel beschriebenen Trade-offs die Diskussion um Komplexität von IT-Architekturen bereichern. [z. B. Sc97, SM03, SWK13] Bisher wird argumentiert, dass unnötige funktionale Redundanz zur Steigerung der Anzahl der Systemelemente und damit zur Komplexität beiträgt. Der Artikel zeigt jedoch auf, dass die Komplexitätsreduktion durch Abbau von funktionaler Redundanz nur bis zu einem gewissen Grad ökonomisch sinnvoll ist.

Praktische Implikationen

Der vorgestellte Ansatz ist ein erster Schritt, um das für IT-Architekten relevante Problem der Ermittlung der kostenoptimalen funktionalen Redundanz in der Zielarchitektur zu untersuchen und Lösungswege aufzuzeigen. Das vorgeschlagene mathematische Modell ist die Grundlage für ein Entscheidungsunterstützungssystem, dessen Prototyp aktuell entwickelt wird. Der Prototyp soll primär die Entscheidungsträger bei der Parametereingabe und Ergebnisinterpretation unterstützen und so die praktische Anwendbarkeit des hier vorgestellten mathematischen Modells vereinfachen. Aus unternehmerischer Sicht gilt es jeweils zu prüfen ob der Mehraufwand der Anwendung des vorgestell-

ten Modells für das zu lösende Problem gerechtfertigt ist – so ist das vorgestellte Optimierungsmodell beispielsweise erst ab einer gewissen Problemgröße und Komplexität der Redundanz- und Kommunikationsbeziehungen sinnvoll. Zudem müssen die Kosten für die präzise Datenerhebung vor dem Hintergrund der möglichen Einsparungen gesehen werden.

Limitationen und weiterer Forschungsbedarf

Da hier der aktuelle Stand eines laufenden Forschungsprojekts berichtet wird, bieten sich zahlreiche interessante Möglichkeiten der Weiterentwicklung. Insbesondere wurde das mathematische Modell noch nicht vollständig in einen Prototypen umgesetzt (die Implementierung läuft jedoch bereits). Zudem wurde der vorgestellte Ansatz bisher nur im Rahmen von einem (jedoch umfangreichen) Projekt evaluiert - hier gilt es weitere Fälle zu betrachten. Der im Aritkel beschriebene Trade-Off fokussiert sich auf den kostenoptimalen Grad an funktionaler Redundanz. Aus der Literatur können jedoch neben dem Kostenaspekt auch weitere, oft strategische und damit schwerer quantifizierbare Vorteile reduzierter Redundanz abgeleitet werden: Eine redundanzfreie IT-Architektur kann zum Beispiel leichter an geplante und ungeplante Änderungen der Geschäftsanforderungen angepasst werden. Daher bietet das vorgestellte Modell lediglich eine Entscheidungsperspektive, die im konkreten Fall durch weitere Betrachtungen zu ergänzen ist. Eine sinnvolle Erweiterung wäre zudem die Berücksichtigung von Unsicherheit bei den Eingabeparametern und eine entsprechend robuste stochastische Optimierung. Auch die statische Betrachtung des Entscheidungsproblems stellt eine Vereinfachung dar. Durch die enge Kooperation mit den Praxispartnern bei der Modellevaluation wurde deutlich, dass insbesondere für Unternehmen in denen jahrelange Konsolidierungsprogramme ablaufen, eine dynamische Betrachtung des Entscheidungsproblems hoch interessant ist - d. h. das Ergebnis der Optimierung sollte neben einer Zielarchitektur auch einen Konsolidierungsplan liefern, der die Reihenfolge der Konsolidierungsschritte festlegt. Diese dynamische Betrachtung ist Gegenstand aktueller Modellerweiterungen. Zudem geht der Artikel von einem binären Redundanzbegriff aus - wie bereits in Kapitel 2.2 diskutiert, sind darüber hinaus in der Realität oftmals auch graduelle funktionale Redundanzen zu finden (z. B. Anwendung A kann 80 % der Funktionen von Anwendung B anbieten). Dies gilt es in entsprechende Modellerweiterungen zu integrieren. In diesem Artikel wurde auf funktionale Redundanz auf Anwendungsebene fokussiert. Gerade da weitere theoretische Implikationen für die Forschung zu EA-Komplexität zu erwarten sind, sollten weitere Typen von Redundanzen (z. B. Datenredundanz) und andere Sub-Systeme der EA (wie zum Beispiel Datenarchitektur und Infrastrukturarchitektur) betrachtet und die entsprechenden Wechselwirkungen in das Modell integriert werden.

Dank

Herzlichen Dank an Markus Weckesser und Dennis Schmidt für die Unterstützung bei der Evaluierung des Modells und ihr hilfreiches Feedback zur Modellformulierung. Marcus Purzer und Dr. Ulf Brusch von der Senacor Technologies AG danke ich für die tatkräftige Unterstützung bei der Instanziierung und die zahlreichen Hinweise zur Praxisrelevanz. Zudem bedanke ich mich bei den zwei anonymen Reviewern für die hilfreichen Hinweise.

Literaturverzeichnis

[Ai09] Aier, S.; Kurpjuweit, S.; Saat, J.; Winter, R.: Enterprise Architecture Design as an Engineering Discipline. In: AIS Transactions on Enterprise Systems, 2009. **1**(1); S. 36-43.

[BB03] Basu, A.; Blanning, R.W.: Synthesis and Decomposition of Processes in Organizations. In: Information Systems Research, 2003. **14**(4); S. 337-355.

[Bu01] Buxmann, P.: Informationsmanagement in vernetzten Unternehmen: Wirtschaftlichkeit, Organisationsänderungen und der Erfolgsfaktor Zeit. Deutscher Universitätsverlag, Wiesbaden, 2001.

[BY06] Boh, W.F.; Yellin, D.: Using Enterprise Architecture Standards in Managing Information Technology. In: Journal of Management Information Systems, 2006. **23**(3); S. 163–207.

[CLT12] Chowanetz, M.; Legner, C.; Thiesse, F.: Integration: An Omitted Variable in Information Systems Research. In: *ECIS 2012 Proceedings.* 2012: Barcelona, Spain.

[EBD11] Espinosa, J.A.; Boh, W.F.; DeLone, W.: The organizational impact of enterprise architecture: a research framework. In: 44th Hawaii International Conference on System Sciences (HICSS), 2011. IEEE.

[GH13] Gregor, S.; Hevner, A.: Positioning and Presenting Design Science Research for Maximum Impact. In: MIS Quarterly, 2013. **37**(2); S. 337-355.

[GP12] Guillemette, M.G.; Paré, G.: Toward a New Theory of the Contribution of the IT Function in Organizations. In: MIS Quarterly, 2012. **36**(2); S. 529-551.

[HF69] Hall, A.D.; Fagen, R.E.: Definition of System. Wiley, New York, 1969.

[IEEE00] IEEE: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. In: IEEE Std 1471-2000, 2000; S. 1-23.

[KWS05] Klesse, M.; Wortmann, F.; Schelp, J.: Erfolgsfaktoren der Applikationsintegration. In: Wirtschaftsinformatik, 2005. **47**(4); S. 259-267.

[LC07] Land, R.; Crnkovic, I.: Software Systems In-House Integration: Architecture, Process Practices, and Strategy Selection. In: Information & Software Technology, 2007. **49**(5); S. 419-444.

[Li06] Lindström, Å.; Johnson, P.; Johansson, E.; Ekstedt, M.; Simonsson, M.: A Survey on CIO Concerns - Do Enterprise Architecture Frameworks Support Them? In: Information Systems Frontiers, 2006. **8**(2); S. 81-90.

[RWR06] Ross, J.W.; Weill, P.; Robertson, D.: Enterprise Architecture as Strategy: Creating a Foundation for Business Execution. Harvard Business Press, Boston, MA, 2006.

[Sc97] Schneberger, S.L.: Distributed computing environments: effects on software maintenance difficulty. In: Journal of Systems and Software, 1997. **37**(2); S. 101-116.

[Si62] Simon, H.A.: The architecture of complexity. In: Proceedings of the American philosophical society, 1962. **106**(6); S. 467-482.

[Si96] Simon, H.A.: The Sciences of the Artificial. MIT Press, Cambridge, MA, 1996.

[SM03] Schneberger, S.L.; McLean, E.R.: The complexity cross: implications for practice. In: Communications of the ACM, 2003. **46**(9); S. 216-225.

[SWK13] Schütz, A.; Widjaja, T.; Kaiser, J.: Complexity in Enterprise Architectures - Conceptualization and Introduction of a Measure from a System Theoretic Perspective. In: *European Conference on Information Systems (ECIS)*. 2013: Utrecht, Netherlands.

[Ta11] Tamm, T.; Seddon, P.B.; Shanks, G.; Reynolds, P.: How Does Enterprise Architecture Add Value to Organisations? In: Communications of the Association for Information Systems, 2011. **28**(1); S. 141-168.

[Wi11] Widjaja, T.: Standardisierungsentscheidungen in mehrschichtigen Systemen: Untersuchung am Beispiel serviceorientierter Architekturen. Gabler, Wiesbaden, 2011.

[Za87] Zachman, J.A.: A Framework for Information Systems Architecture. In: IBM Systems Journal, 1987. **26**(3); S. 276-292.

Anhang

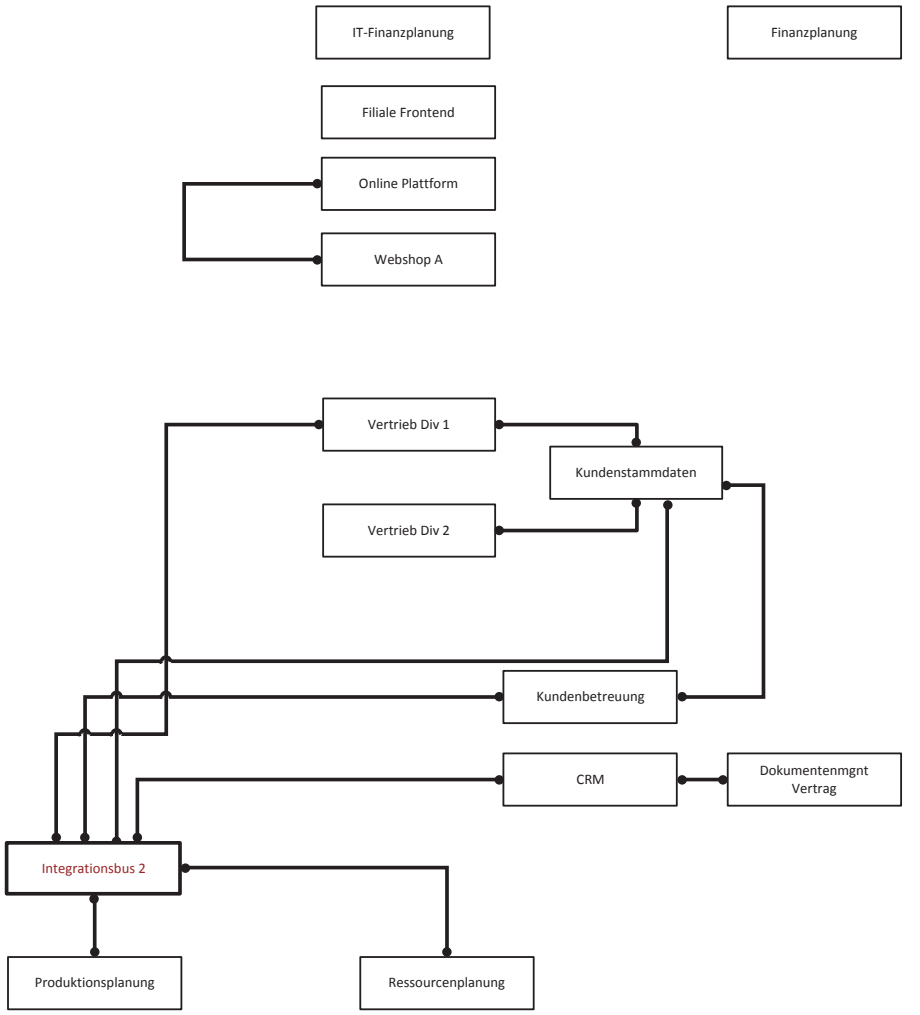

Abbildung 5: Schematische Darstellung der Soll-Architektur